NATURAL LANGUAGE PROCESSING (NLP) - ELABORAZIONE DEL LINGUAGGIO NATURALE

È una branca dell’AI ed è chiamata ***NLP (Natural Language Processing)****;* è il processo di trattamento automatico delle informazioni scritte o parlate in una lingua naturale mediante un calcolatore.

Essa è composta da 2 componenti principali:

***Natural Language Understanding (NLU)***: conversione del linguaggio naturale in linguaggio artificiale tramite sistemi di riconoscimento vocale.

***Natural Language Generation (NLG)***: generazione del linguaggio naturale da parte del calcolatore.

DEEP LEARNING – APPROFONDIMENTO PROFONDO

Il Deep learning è la branca dell’AI che si occupa principalmente del *training* delle reti neurali. Nel campo dell’apprendimento automatico, una rete neurale artificiale è un modello matematico/informatico di calcolo basato sulle reti neurali biologiche; è costituita da un gruppo di “neuroni artificiali”. In pratica una rete neurale artificiali si ispirano alle reti neurali biologiche (esempio la rete cerebrale del nostro cervello), cercando una emulazione tramite algoritmi (ovviamente la complessità, al giorno d’oggi, è molto inferiore). Le reti neurali prendono decisioni in autonomia migliorandosi con il tempo, senza necessità dell’intervento umano.

LUIS (Language Undestanding Intelligent Service): servizio di machine learning che permette di integrare il bot a tutti i servizi di NLP; progettato per estrarre le informazioni rilevanti durante la comunicazione con l’utente, capire le richieste (*intents*) e identificarne le parole chiave all’interno di una frase (*entities*). Per quanto riguarda il costo viene messo a disposizione un piano gratuito soggetto a limitazioni relative al numero di richieste al mese (10∙000 richieste al mese e massimo 5 richieste al secondo)

Funzionamento Microsoft Bot Framework

Turni e contesti di attivazione

La ricezione di una attività da parte dell’utente a la relativa elaborazione attraverso il bot è chiamata **turno**. Quando viene ricevuta una attività viene creato un **contesto di attivazione** che esiste fino alla terminazione del turno; esso può essere utilizzato per archiviare informazioni che potrebbero essere necessarie più avanti nel turno.

Ogni attività ricevuta dall’applicazione bot viene passata ad un bot adapter che passa le info relative ad essa alla logica del bot e risponde all’utente. Una attività è formattata in codice JSON, il quale viene deserializzato e passato nel payload delle chiamate Post per poi essere utilizzato dal bot.

**Bot adapter**

Quando il bot riceve una attività, l’adapter esegue il wrapping di tutti gli elementi di quell’attività, crea un contesto di ambiente per il turno, lo passa alla logica applicativa del bot e si occupa di inviare le risposte generate dal bot sul canale di comunicazione.

L’adapter autentica ogni attività in ingresso usando l’intestazione *Authentication* dalla richiesta REST e usa le credenziali dell’applicazione per autenticare le attività in uscita per l’utente. L’autenticazione del servizio Bot Connector usa token *Bearer JWT (JSON Web Token)* insieme all’ID dell’app Microsoft e alla password dell’app Microsoft che Azure crea automaticamente quando si registra un bot (in locale non è necessario).

**Contesto di Turno e stato della conversazione**

*(https://docs.microsoft.com/it-it/azure/bot-service/bot-builder-howto-v4-state?view=azure-bot-service-4.0&tabs=csharp)*

Quando un adapter riceve un’attività, genera un contesto di turno che immagazzina info sull’attività in ingresso (es: mittente, destinatario, canale di comunicazione ecc). L’adapter passa il contesto al bot che elabora l’attività.

La gestione dello stato della conversazione è strutturata sotto forma di **proprietà di stato**, ovvero coppie di chiave-valore alle quali il bot può accedere; ciò permette di salvare direttamente le istanze delle classi in modo persistente senza preoccuparsi dell’implementazione sottostante.

Le classi messe a disposizione dall’SDK permettono diverse tipologie di gestione dello stato, in particolare:

* *UserState:* è disponibile in ogni turno in cui il bot conversa con quell’utente in quel canale. Chiave identificativa formata da <*ChannelId, FromId>*
* *ConversationState:* disponibile in qualsiasi turno di una conversazione indipendentemente dall’utente (conversazione di gruppo). <*ChannelId, ConversationId>*
* *PrivateConversationState:* l’ambito della conversazione dipende dalla singola conversazione con un dato utente. < *ChannelId, FromId, ConversationId* >

*BasicBotAccessor* viene creata come singleton durante la configurazione dalla classe *Startup* e viene passata alla classe *BasicBot* ad ogni turno*;* questo *accessor* mi permette di accedere allo stato della conversazione offrendo la possibilità ai vari moduli di leggere/scrivere i vari campi. Questi campi implementano l’interfaccia *IStatePropertyAccessor*.

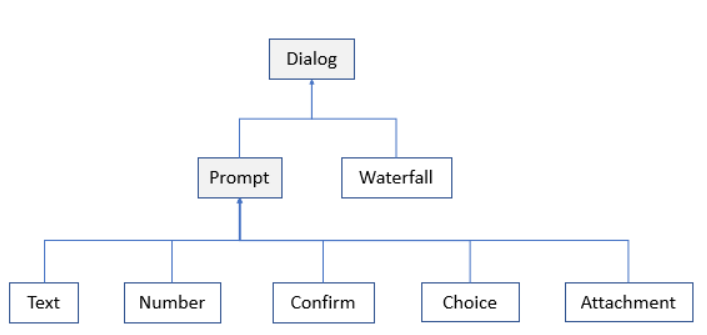
Dialoghi

Elaborano le attività in ingresso e generano risposte in uscita. Le istanze di dialogo vengono disposte in uno stack che rimane persistente tra i vari turni e non ha vincoli temporali (può durare diversi giorni).

Un dialogo implementa 3 funzioni principali:

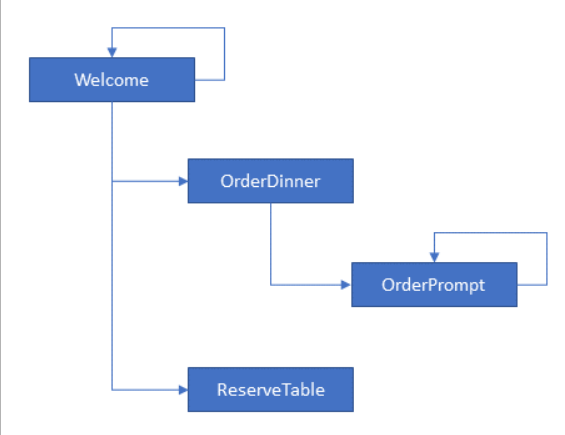
* BeginDialog
* ContinueDialog
* ResumeDialog
* EndDialog
* CancellAllDialog
* ReplaceDialog

La classe *DialogContext* e i dialoghi interagiscono per scegliere il dialogo appropriato per gestire l’attività. La classe *DialogContext* collega lo stack del dialogo persistente, l'attività in ingresso e la classe *DialogSet*. Un elemento *DialogSet* contiene dialoghi che possono essere chiamati dal bot.

Dialoghi e prompt a cascata

La libreria *Dialog* contiene un set di tipi ***Prompt*** (richieste), le quali vengono usate all’interno del “dialogo a cascata” per richiedere i vari input agli utenti.

Un elemento *WaterfallDialog* indica i vari “step” del dialogo.



È possibile creare rami per indirizzare la conversazione; il contesto di dialogo mantiene uno stack di dialoghi, i quali possono essere aggiunti, rimossi o sostituiti. Il dialogo in cima allo stack viene detto *dialogo attivo*.

Dialoghi Componenti

A volte si vuole scrivere un dialogo che possa essere usato in diversi scenari. Un esempio potrebbe essere un dialogo relativo agli indirizzi che chiede all'utente di fornire valori per via, città e codice postale.

L'elemento *ComponentDialog* fornisce un livello di isolamento perché ha un elemento *DialogSet* separato ed evita conflitti di nomi con il padre che contiene il dialogo creando il proprio runtime interno.

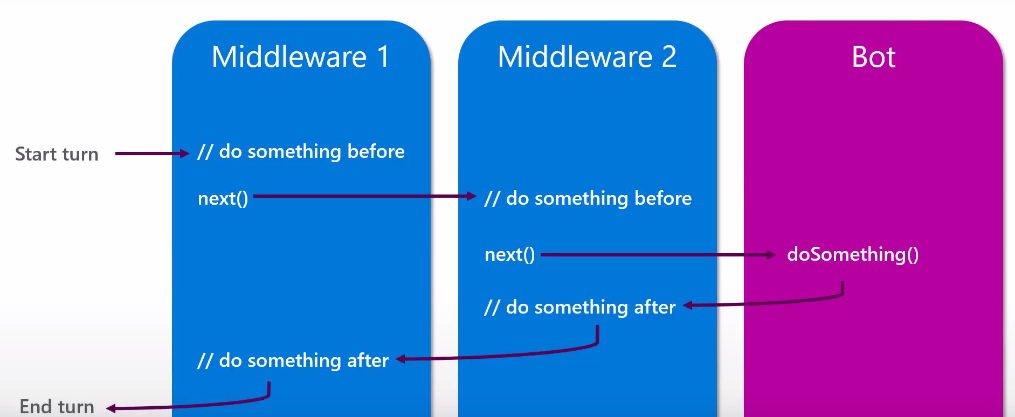
Stato dei Dialoghi

*Accessor* è un componente utilizzato dai dialoghi, che offre metodi di Get, Set e Delete per l’accesso ai dati persistenti.

Middleware

È una classe che si trova tra l’Adapter e la logica del bot. L'SDK consente di scrivere il proprio middleware o di aggiungere componenti riutilizzabili del middleware creati da altri utenti. Tutte le attività che entrano o escono dal bot passano attraverso il middleware. Il middleware offre altre opportunità per interagire con il flusso di conversazione degli utenti, sia prima che dopo l'elaborazione di ogni turno della conversazione. Il middleware consente anche di archiviare e recuperare le informazioni relative alla conversazione.

**Pipeline middleware del Bot**

Next(): invoca il prossimo componente nella pipeline.

Programmazione asincrona

I metodi asincroni sono definiti con la parola chiave async e il nome del metodo dovrebbe, per convenzione, terminare con la parola *Async*.

Il valore restituito è:

* Task<TResult> se nel metodo è presente un’istruzione return dove l’operando è di tipo TResult
* Task se nel metodo non è presente un'istruzione return oppure è presente un'istruzione return senza l'operando.
* Void se si sta scrivendo un gestore eventi asincrono.

Funzionamento: Il metodo include in genere almeno un'espressione await, che contrassegna un punto in cui il metodo non può continuare fino a quando l'operazione asincrona attesa non sia completata. Nel frattempo, il metodo viene sospeso e il controllo ritorna al chiamante del metodo.

Note implementative:

Per ogni caso d’uso sviluppare app LUIS separatamente che verranno gestite con il tool dispatch.

La gestione degli intents/entities può essere gestita tramite la pipeline middleware; l’istanza del bot esegue le solo le richieste; configurare i middleware in Startup.cs

Tools

Azure CLI (Command Line Interface): strumento che permette la gestione delle risorse/servizi di Microsoft Azure da terminale.

Ci permette di gestire tutti i tools per lo sviluppo del Bot:

-CHATDOWN: Prototype mock conversations in markdown and convert the markdown to transcripts you can load and view in the new V4 Bot Framework Emulator.

-msbot: crea e gestisce le connessioni ai vari servizi del bot (creazione del file BotConfiguration.bot)

-LUDOWN: tool che converte i file *.lu* in file JSON, i quali possono essere importati per creare automaticamente applicazioni LUIS con gli intent/entities specificati (alternativa al portale luis).

-LUIS: applicazione luis

-QnAMarker: permette la creazione di un servizio di domande e risposte (permette di non doverle gestire con intent rendendo l’app LUIS più snella)

-DISPATCH TOOL: permette la gestione automatica dei vari servizi connessi al bot

Flusso di sviluppo:

Documentazione per i CLI Tools: [*https://githu-b.com/Microsoft/botbuilder-tools*](https://githu-b.com/Microsoft/botbuilder-tools). Per vedere tutte le opzioni dei comandi di esempio mostrati successivamente consultare la documentazione.

1) Creare i file di testo *nome.lu*, ognuno dei quali specifica *Intents ed Entities* di un modulo LUIS

2) Creare file JSON tramite il tool *LuDown*: (*ludown parse ToLuis --in ludown-file-name.lu)*

3) Creare i moduli LUIS dal portale Luis.ai di Microsoft: [*https://www.luis.ai/home*](https://www.luis.ai/home). I moduli vanno pubblicati per far si che possano essere raggiunti. (per importare una nuova versione di un modello LUIS modificare il campo version nel file json generato con ludown, altrimenti il portale “lo rifiuta”)

4) Creare il file *name.bot* tramite il tool *MsBot*: *(msbot init --name TestBot --endpoint http://localhost:9499/api/messages)*

5) Creare il Dispatch Model utilizzando il file *name.bot.* [*https://github.com/Microsoft/botbuilder-tools/tree/master/packages/Dispatch*](https://github.com/Microsoft/botbuilder-tools/tree/master/packages/Dispatch) *-> Vedi paragrafo* ***Create bot dispatch using bot file***

6) Configura il codice sorgente seguendo la documentazione e il progetto *NLP-With-Dispatch-Bot* della solution *csharp\_dotnetcore*

In caso di modifica di un modulo LUIS:

* *ludown parse ToLuis --in ludown-file-name.lu*
* *modifica nel file \*.json creato il campo* “versionId” (es: "versionId": "0.5")
* *modifica il file \*.bot l’id del servizio modificato coincidente con l’id specificato nel punto sopra*
* importare sul portale luis la nuova versione del modulo .json
* train e public del modulo luis
* Aggiornare e ripubblicare dispatch dal portale LUIS.
* Comando 1: dispatch refresh --bot c:\src\bot\testbot.bot --secret <your\_bot\_file\_secret>
* Comando 2: dispatch eval --luisSubscriptionKey <auth\_key sul portale luis> --luisSubscriptionRegion westus

Se invece aggiungo un modulo nuovo usare msbot per aggiungere quel servizio luis.

Telegram

Iniziare conversazione con telegram: https://telegram.me/<bot name>

Posso aggiungere parametri alla chiamata (start/startgroup): es: https://telegram.me/<bot name>?start=<value>

Prima bozza indice

INDICE

Introduzione

*[introduzione sulle chatbot e applicazione sanitaria] – niente di specifico a livello tecnico -> cerca articoli attualità (analisi di mercato)*

1 Ambiente di riferimento (panoramica degli argomenti cardine che si andranno a toccare)

-AI, NLP, Deep learning

-chatbot

-assistente sanitario

-Onit s.r.l.

-Iter attuale per prenotazione (statistica delle persone che prenotano di “persona”)

Problema e Obiettivo del progetto/tesi

-descrizione dettagliata di cosa si desidera sviluppare e qual è lo scopo della tesi/progetto

2 Analisi requisiti

-descrizione dettagliate di tutte le funzionalità (specificare quali sono funzionali e non funzionali, diagrammi UML sui casi d’uso)

-varie criticità (es: trattando dati sanitari considerare aspetti come: privacy, locazione dei datacenter, autenticazione, criptazione del canale…), eventuali problematiche/errori che si possono riscontrare, scalabilità del bot (un singolo bot che soddisfa tutte le richieste (collo di bottiglia) oppure bot dedicato a singolo utente)

-competitor sul mercato

3 Progettazione (descrivi le scelte progettuali a fronte delle analisi preliminari eseguite)

a) COSA FARE:

- framework utilizzati e non (descrivere svariati framework presi in esame e riportare le considerazioni che implicano le scelte progettuali fatte – es: perché Microsoft API piuttosto che Amazon API)

-sistema di comunicazione (app proprietarià android (registrazione e autenticazione), app preesistente (pin-telegram), browser …) => dimostrativo per lo sviluppo della tesi

-onHealth API (studia doc)

-macchine a stati per realizzazione del bot

b) COME FARE (linee guida per una buona realizzazione):

-modello a cascata (Analisi requisiti -> progettazione -> realizzazione -> testing e deploy), modulare (eventuali nuove specifiche richiederanno modifiche ad una parte minima del progetto)

4 Realizzazione

Percorso di sviluppo: descrivere passo a passo gli step

*Es: step1: realizzazione dell’infrastruttura del bot; step2: realizzazione app android; step3 …*

Eventualmente mostrare snapshot durante l’utilizzo

5 Testing ed (eventualmente) Deploy

6 Conclusioni

-raggiungimento obiettivi

-conoscenze acquisite

-considerazioni personali

2 Analisi dei requisiti

Descrizione

L’obiettivo della tesi è la progettazione di una chatbot, ovvero un sistema di messaggistica istantanea connesso ad un assistente virtuale messo a disposizione dell’utente h24, che trova applicazione in ambito sanitario. L’assistente sfrutta le tecnologie di *Natural Language Processing* che permettono all’utente la comunicazione con il sistema attraverso il linguaggio naturale, facilitando tutte le operazioni *[CASI D’USO]* di prenotazione di visite mediche, gestione della refertazione, sistema di promemoria, richieste di informazioni legate alle visite specialistiche…

CASI D’USO

* Prenotazione/spostamento/revoca appuntamenti (principale)
* Promemoria/sistema di notifica (principale)
* Refertazione (secondario)
* Scheda del benessere: richiesta di info rispetto ad una visita specialistica (cos’è, come si svolge, cosa fare/non fare, ogni quanto fare una determinata visita, pagamento ticket)

Primo capitolo

Quadro del problema e introduzione del progetto/motivazioni.

Secondo

Cosa sono chatbot e come si applicano