

POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

**Assistente Conversazionale e
Multimodale per Veicoli Connessi**



Relatori

Fulvio Corno

Luigi De Russis

Alberto Monge Roffarello

Candidato

Eugenio Gallea

Aprile 2019

Indice

Elenco delle figure	4
1 Introduzione	6
2 Soluzioni sul mercato e confronto	13
2.1 Seat e Amazon Alexa	13
2.2 Mercedes e “Hey Mercedes”	14
2.3 BMW e “Hey BMW”	16
2.4 Analisi del confronto	16
3 Casi d’uso	17
3.1 Idea generale	17
3.2 Categorie di casi d’uso considerati	18
3.2.1 Da guidatore: Comando	19
3.2.2 Da guidatore: Richiesta informazioni	19
3.2.3 Da veicolo: Avvisi	19
3.2.4 Da veicolo: Alert critici	20
3.2.5 Motivazione della scelta	20
3.3 Casi d’uso implementabili	20
3.3.1 Da guidatore: Comando	21
3.3.2 Da guidatore: Richiesta informazioni	23
3.3.3 Da guidatore: Richiesta di informazioni (derivate)	25
3.3.4 Da auto: Avvisi	27
3.3.5 Da auto: Alert critici	29
4 Progettazione del sistema	31
4.1 Architettura del sistema	31
4.1.1 Voice Recognition & Speech Synthesis	32
4.1.2 Conversational Engine & Intelligence	33
4.1.3 Car Simulator & HMI Simulator	36

5	Implementazione	39
5.1	Architettura e comunicazione fra i componenti	39
5.2	Interazione utente-simulatore	41
5.3	Componenti nel prototipo	43
5.3.1	Assetto Corsa	43
5.3.2	DialogFlow	43
5.4	L'applicazione sul veicolo	49
5.4.1	Modulo Server	51
5.4.2	Modulo per lettura della telemetria	52
5.4.3	Modulo Client	54
5.4.4	Modulo per interazione con cruscotto integrativo	56
5.4.5	Modulo per interazione con la base dati	57
5.5	Dashboard	58
5.6	Casi d'uso e interazione dei componenti	61
5.6.1	Da guidatore: Comando	61
5.6.2	Da guidatore: Richiesta di informazioni (derivate)	64
5.6.3	Da auto: Avvisi	66
6	Risultati	69
6.1	Specifiche degli elaboratori	69
6.1.1	Elaboratore simulatore	69
6.1.2	Elaboratore assistente	69
6.2	Test funzionale	71
6.2.1	Risultati test funzionale	71
6.3	Latenza di risposta dell'assistente	78
6.4	Considerazioni finali	80
6.4.1	Limite nella gestione dei contesti	80
6.4.2	Interazione con il guidatore	81
7	Conclusioni	83
7.1	Lavori futuri	84
	Bibliografia	87

Elenco delle figure

2.1	Visione di un cruscotto su un veicolo Seat [4]	13
2.2	Visione dello schermo con evidenziata l'opzione per attivare Amazon Alexa [5]	13
2.3	Visione lato guidatore di un cruscotto Mercedes [7]	14
2.4	Visione lato guidatore di un cruscotto Mercedes [8]	14
3.1	Diagramma che descrive come avviene l'interazione con l'assistente	17
4.1	Architettura del sistema e interazione fra i componenti.	32
4.2	Risultati del confronto svolto tra tre assistenti conversazionali presenti sul mercato: DialogFlow, Wit.ai, IBM Watson.	35
5.1	Rappresentazione di come avviene la comunicazione tra i vari componenti del prototipo e la loro suddivisione.	40
5.2	Figura che evidenzia il modulo contenente l'Intelligence del sistema e Dashboard integrata.	42
5.3	Figura che evidenzia i moduli per Car Simulator & HMI Simulator.	42
5.4	Visione della sezione Fulfillment nella console di DialogFlow	44
5.5	Visione della sezione Entities del prototipo nella console di DialogFlow	44
5.6	Visione della sezione Entities del prototipo nella console di DialogFlow	45
5.7	Visione della sezione Intent del prototipo nella console di DialogFlow	46
5.8	Visione di uno specifico intento definito per il prototipo.	47
5.9	Visione della sezione Fulfillment nella console di DialogFlow	47
5.10	Visione di un intento attivato da un evento.	48
5.11	Visione di come inserire all'interno del contesto generato da un intento i dati provenienti dall'evento.	49
5.12	Visione dell'organizzazione dei packages all'interno dell'applicativo.	50
5.13	Visione dell'organizzazione interne del package server.	51
5.14	Visione dell'organizzazione del package ac_interaction.	52
5.15	Visione dell'organizzazione del package client.	54
5.16	Visione dell'organizzazione del package database.	57
5.17	Visione della dashboard come appare all'utente.	59

5.18	Visione dell'organizzazione dell'applicazione per la dashboard integrativa.	60
5.19	Architettura del sistema con evidenziati i componenti utilizzati per questo caso d'uso.	62
5.20	Architettura del sistema con evidenziati i componenti che agiscono nella fase di ricezione e aggiornamento dei dati sul veicolo.	65
5.21	Architettura del sistema con evidenziati i componenti che agiscono nella fase di riconoscimento del cambiamento di stato di un indicatore.	66
6.1	Specifiche dell'elaboratore utilizzato per il deploy dell'assistente.	70
6.2	Specifiche dell'elaboratore utilizzato per il deploy dell'assistente.	70
6.3	Specifiche della velocità di trasmissione della scheda di rete dell'elaboratore.	71
6.4	Figura contenente un esempio di conversazione con cambio contesto.	77
6.5	Figura contenente un esempio di errore durante la conversazione iniziata dall'assistente.	79
6.6	Figura contenente un esempio di errore durante la conversazione iniziata dal guidatore.	79
6.7	Velocità della connessione di rete usata durante la misurazione delle latenze.	80

Capitolo 1

Introduzione

Sempre di più, al giorno d'oggi, si è alla ricerca di tecnologie intelligenti che sappiano facilitare le azioni quotidiane dell'utilizzatore. Per capire meglio: consideriamo un utente medio intenzionato a comprare un qualcosa di *smart*, egli/ella non è semplicemente interessato a un computer, un telefono, un'automobile in grado di fare tante operazioni molto complicate, risolvere problemi molto difficili o quant'altro ci si possa aspettare da una tecnologia dotata di capacità di elaborazione, bensì ciò che viene richiesto è un qualcosa in grado di **semplificare** le azioni quotidiane dell'utilizzatore.

Quello che succede nella stragrande maggioranza dei casi, però, è che un utente, non conoscendo appieno tutte le potenzialità dell'accessorio da lui acquistato, non riesce a sfruttarlo al meglio. Sono rarissimi i casi in cui una persona, dopo avere acquistato, ad esempio, un nuovo smartphone, vada a leggere la lista completa di tutte le sue funzionalità. Ancora meno frequenti sono i casi in cui un utente, dopo l'acquisto di un'auto nuova con moltissimi accessori e comodità vada a leggerci di "sua spontanea volontà" il manuale d'istruzioni, o addirittura il manuale del veicolo per capire il significato di tutti i pulsanti, spie o qualsiasi altra cosa sia presente sul cruscotto. I prodotti sul mercato commerciale, sono pensati per essere *plug and play*. A volte, però, avere la possibilità di sfruttare al massimo le potenzialità dello strumento, può ritornare molto utile, come anche può risultare necessario, se non fondamentale, conoscere il significato di particolari segnali che ci vengono inviati dal dispositivo. Nell'esempio della macchina citato in precedenza, il guidatore può incorrere nella situazione in cui vedendo una spia accendersi sul cruscotto, non presti particolare attenzione alla stessa conoscendone in parte o non ignorandone totalmente il significato. Si consideri un caso apparentemente innocuo come, ad esempio, l'accensione della spia del liquido di raffreddamento. Essa può avere diversi significati: il primo, più semplice, è quello di indicare che il liquido di raffreddamento non ha ancora raggiunto la temperatura adeguata; in alcuni casi, però, se l'accensione di tale spia avviene insieme all'accensione di altri indicatori, il suo significato cambia, infatti potrebbe indicare la presenza di un guasto all'interno

del motore. Di ciò l'utente non ne può essere a conoscenza senza andare a leggere il manuale completo del veicolo. Da questo punto di vista viene molto in aiuto l'avanzamento tecnologico degli ultimi tempi, soprattutto a livello *software*, che ha semplificato enormemente l'interazione uomo-macchina. Un utilizzatore medio, non è più obbligato a interagire con la macchina nel "linguaggio" della macchina, bensì è quest'ultima a porre l'interazione tra i due su un livello umano.

Ecco, quindi, l'entrare in gioco degli *assistenti conversazionali*, che non sono altro che programmi software, decorati con algoritmi di intelligenza artificiale, in grado di sostenere una conversazione con un utente finale e di capirne eventuali richieste. L'applicazione di questi assistenti ha toccato molti settori. Primo su tutti quello degli smartphone, chiunque, ormai è familiare con Siri, Google Assistant e ne è un utilizzatore quotidiano. Un altro campo in cui tali assistenti hanno preso piede è quello della domotica, con Alexa e Google Home, ormai in grado di rispondere a qualsiasi richiesta, partendo dalla più semplice come per esempio "Quali sono le previsioni del tempo per domani?", fino ad arrivare alla possibilità di regolare il colore delle luci nella stanza. Altri tipi di chatbot (all'inglese), sono stati impiegati per la gestione dei messaggi su pagine Facebook, oppure su canali e chat Telegram. Insomma, il loro utilizzo è molto diffuso e il principale motivo di questo è quello che sono molto **facili e intuitivi** da usare.

Appurata la loro effettiva utilità e semplicità di utilizzo, la vera domanda che bisogna porsi è il perchè di un così massiccio investimento da parte di molte compagnie nello sviluppo di tali assistenti. Il motivo è abbastanza semplice: come anticipato in precedenza, un'interfaccia utente di tipo conversazionale, permette all'utilizzatore di avere il privilegio di interagire con un computer senza sapere alcun linguaggio di programmazione particolare, bensì utilizzando semplicemente termini umani [1]. Il tipo di comunicazione quindi è stato completamente spostato da come era qualche tempo fa. Non è più necessario l'inserimento di parole chiave, il cliccare icone o pulsanti particolari per chiedere l'esecuzione di azioni specifiche. Basta la **voce**.

Proprio per questo motivo, molte case automobilistiche hanno cominciato utilizzare questo tipo di interazione nei loro prodotti e a integrare assistenti conversazionali all'interno del veicolo. Il guidatore ha, così, la possibilità di poter chiedere l'esecuzione di azioni specifiche all'assistente conversazionale in maniera immediata, parlando. Non ci fosse stata questa possibilità tale gesto avrebbe richiesto il doversi distrarre dalla guida.

Negli ultimi anni, quindi, è diventato sempre più comune imbattersi in automobili che permettano di comandare telefonate sul cellulare utilizzando il veicolo collegato ad esso via Bluetooth, impostare una destinazione per il navigatore a voce e molto altro. Le soluzioni che sono state messe sul mercato in prodotti commercializzati, delle quali si è parlato più recentemente sono:

- l’integrazione nei veicoli Seat di Amazon Alexa, un’idea innovativa, ma che consiste semplicemente nell’inserimento dell’assistente Amazon all’interno di un’auto Seat, senza una vera e propria integrazione tra i due;
- Mercedes con il suo assistente (proprietario, al contrario del caso precedente) “Hey Mercedes”, che già è un enorme passo avanti rispetto alla proposta di Seat, ma che comunque presenta ancora qualche limite, discusso più nel dettaglio nel capitolo 2;
- BMW, anche questa casa con il suo assistente proprietario, “Hey BMW”, che risulta essere un passo in più avanti a quello di Mercedes e molto più integrato con il veicolo rispetto a quello Seat. Anch’esso, però, come “Hey Mercedes”, presenta alcuni limiti sostanziali, anch’essi discussi più nel dettaglio nel capitolo 2;

Tutte queste sono soluzioni molto utili e migliorano enormemente l’esperienza di guida del veicolo, ma è proprio questo il problema: **migliorano**, non **semplificano**. Infatti davanti ai cruscotti di questi veicoli, il guidatore trova una miriade di pulsanti, pulsantini, leve e touchscreen, che anziché facilitarlo nell’utilizzo dell’assistente, lo confondono e in ultimo luogo lo **distraggono** dal suo compito principale: **guidare**. Questo è solo uno dei problemi principali riscontrabili in questo tipo di assistenti, un utente dietro al volante, non deve mai togliere lo sguardo dalla strada al fine di potere attivare l’assistente o selezionare un sottomenù per ottenere particolari informazioni. Il suo compito principale è quello della guida e l’assistente deve essere di aiuto in questo senza risultare troppo invasivo.

Un altro limite di tali prodotti risiede nel fatto che queste soluzioni sono tutte per assistenti **reattivi**, ossia la conversazione è sempre iniziata dall’utente, che deve chiedere informazioni, impartire ordini o qualsiasi altra azione si voglia effettuare. Tutto ciò può sembrare legittimo, fino a che non si vanno a considerare, ad esempio, situazioni di guasti al motore. In questi casi, spesso, l’utente non è in grado di accorgersene da solo, di conseguenza l’assistente dovrebbe avere la capacità di avvisare la presenza di tali problemi nel veicolo. Da qui la necessità di creare un assistente non più semplicemente reattivo, bensì **proattivo**. In altre parole, l’applicazione non solo è in grado di rispondere e sostenere una conversazione iniziata dal guidatore, ma anche di avviarla lei stessa, nel caso si verificano situazioni particolari.

L’obiettivo di questa tesi, quindi, è quello di creare un assistente **proattivo** e **multimodale** (ossia in grado di interagire con l’utente in diversi modi, nel caso specifico tramite interazione vocale e visualizzazione dei dati su video), che abbia una forte integrazione con il veicolo, ma che soprattutto non sia invasivo e **non distra** l’utente. Un’applicativo con queste proprietà sarà, quindi, in grado di avvisare il guidatore di eventuali problemi al motore o altre criticità iniziando lui stesso la conversazione. Questa è forse la capacità da prioritizzare in un assistente

perchè evita al guidatore di scoprire problemi al veicolo quando ormai è troppo tardi, prevenendo possibili situazioni di pericolo. È molto importante però mantenere una certa continuità con gli assistenti già presenti sul mercato e quindi evitare che l'aggiunta di queste proprietà non elimini dall'assistente quelle capacità già presenti e testate nelle soluzioni commerciali citate in precedenza. Ad esempio l'assistente deve mantenere la possibilità di soddisfare richieste dell'utente, spastandone però l'attenzione da semplici comandi come "Imposta tale destinazione sul navigatore" a comandi più inerenti al veicolo in sé, come ad esempio "Accendi i fendinebbia". Lo stesso principio viene applicato anche a richieste di informazioni su accessori a bordo del veicolo.

In letteratura, questo tipo di interazione bidirezionale tra veicolo e guidatore è già stata esplorata e considerata, ma non implementata in una situazione reale. È possibile trovare vari esempi di studi a proposito di come il linguaggio naturale possa costituire una facilitazione per il guidatore nell'utilizzo delle potenzialità dell'assistente, ma anche del veicolo stesso, con il minimo sforzo. Ad esempio in uno studio condotto da David R. Large et al. [2] (2017) viene sostenuto che l'interazione con il veicolo in termini colloquiali "[...] *innalza lo stato sociale dell'assistente conversazionale, ponendolo allo stesso livello dell'utente [...]*". Questo avvalorava il fatto che più la conversazione tra chatbot e utente è "umana" e meno egli/ella, trovandosi a proprio agio nell'interloquire con l'assistente, è portato/a a distrarsi dalla guida.

Inoltre moltissime volte capita che l'utilizzatore del veicolo non conosca tutte le funzionalità di esso e neanche conosca il significato di tutte quante le spie del cruscotto. Per questo motivo, l'assistente deve essere in grado di aiutare l'utente nel comprendere il significato di un simbolo, capire il funzionamento di un accessorio o qualsiasi altra cosa riguardi l'effettivo utilizzo del veicolo. Questo è possibile solamente grazie a una forte interazione tra applicazione e mezzo [3].

Al fine di creare un assistente poco distraente e differente da quelli attualmente in commercio, si è cercato di tenere minimale e poco invasiva la parte di interazione con l'utente, limitandola solamente alla comunicazione vocale (attraverso microfono) e visiva (visualizzazine su display di informazioni minime sullo stato del veicolo). L'architettura del sistema risulta, di conseguenza, molto semplice e composta da tre parti principali: un simulatore di guida (che ha la funzione di rappresentare il veicolo dal quale estrarre dati, non avendone uno reale a disposizione), l'applicazione contenente la logica e una dashboard integrativa. Il simulatore fornisce dati all'applicazione, quest'ultima invia i dati ricevuti a un cruscotto integrativo per la visualizzazione degli stessi ed è responsabile dell'ascolto delle richieste dell'utente e della produzione di una risposta tramite una sua logica interna.

Lo sviluppo è stato suddiviso in diverse fasi, partendo dall'analisi del problema

nelle soluzioni attualmente in commercio, fino ad arrivare al testing del prototipo. Gli step nei quali è possibile suddividere il processo sono i seguenti:

1. analisi e confronto delle soluzioni commerciali attualmente presenti sul mercato. Come citato in precedenza, tra le soluzioni presentate da Seat, Mercedes, BMW e quella proposta in questa tesi è possibile notare alcune differenze sostanziali. Infatti il nostro assistente al contrario di ciò che ora si trova sul mercato punta ad avere una forte integrazione con il veicolo e ad essere proattivo;
2. analisi, confronto e scelta di un'adatta *conversational engine*, grazie alla quale sarà possibile definire come l'assistente governerà la conversazione;
3. scelta di un valido simulatore di guida per emulare una situazione reale;
4. decisione di casi d'uso al fine di definire dei test funzionali per provare le effettive capacità dell'assistente;
5. design dell'architettura del prototipo;
6. sviluppo del prototipo con annessa scelta dei servizi/libreria utilizzabili;
7. test funzionali dell'assistente con i casi d'uso decisi al punto 3.

Essendo il prototipo inserito in un contesto di veicoli connessi, durante sviluppo, si è cercato di massimizzare l'utilizzo di *cloud services* per lo svolgimento di determinate funzioni. Infatti come verrà spiegato esaustivamente nel capitolo 5 sono stati sfruttati servizi cloud sia come conversational engine, sia per tradurre l'input vocale dell'utente in scritto (speech-to-text) che il contrario (text-to-speech). Questo fattore ha influito particolarmente nel processo di organizzazione dell'architettura che ha portato alla creazione dei diversi moduli di cui è composto il prototipo, ma in particolare l'applicativo installato sul veicolo, discussi più nel dettaglio nel capitolo 5.

Nella valutazione funzionale sono stati ottenuti buoni risultati: l'assistente riesce a gestire correttamente il 100% delle conversazioni modellate nei casi d'uso definiti, accettandone anche alcune variazioni (ad esempio sul come è formulata una domanda oppure su quale sinonimo di una certa parola viene utilizzato). Inoltre l'applicazione è in grado di cambiare contesto di conversazione anche se si è partiti da un contesto diverso.

La presentazione del lavoro svolto con questa tesi, verrà organizzata in questo documento come segue:

- nel capitolo 2 verranno discusse più approfonditamente le differenze tra le soluzioni commerciali di Seat, Mercedes e BMW per i loro assistenti conversazionali e la soluzione sviluppata in questo progetto;

- nel capitolo 3 saranno presentati i casi d'uso sui quali sono stati basati i test funzionali per provare le effettive capacità dell'assistente;
- nel capitolo 4 verrà spiegata l'architettura generale del sistema;
- nel capitolo 5, riprendendo l'architettura presentata al capitolo precedente, verranno esposte le effettive tecnologie utilizzate;
- nel capitolo 6 verranno mostrati i risultati ottenuti dai test funzionali e espressi possibili lavori futuri di espansione dell'assistente.

Capitolo 2

Soluzioni sul mercato e confronto

In questo capitolo verrà descritto il lavoro che è stato svolto durante quella che è da considerarsi la prima fase della tesi. Sono state raccolte informazioni sulle soluzioni attualmente in commercio per assistenti conversazionali a bordo di veicoli. Questa analisi è stata necessaria per potere focalizzare il lavoro di sviluppo sulle caratteristiche mancanti negli assistenti attuali.

Di seguito verranno presentate in maniera più dettagliata le soluzioni più vendute sul mercato spiegando le principali differenze con il prototipo creato. In particolare verranno elencati gli esempi di *Seat*, *Mercedes* e *BMW*.

2.1 Seat e Amazon Alexa



Figura 2.1: Visione di un cruscotto su un veicolo Seat [4]

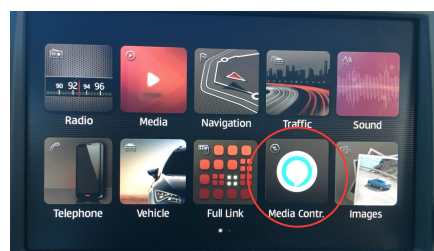


Figura 2.2: Visione dello schermo con evidenziata l'opzione per attivare Amazon Alexa [5]

Lo scorso anno, Seat ha presentato l'integrazione dell'assistente Amazon Alexa sui suoi veicoli, prima in Germania e Regno Unito, poi dal 12 Novembre 2018 [4] anche in Francia, Spagna e Italia. Innanzitutto è sicuramente un **vantaggio** avere

all'interno dell'auto un assistente vocale, con il quale, come dice il nome stesso, si può interagire utilizzando la voce e di conseguenza *evitando* di distrarsi dalla guida. Ecco quello che l'assistente introdotto da Seat **può** fare [6]:

1. gestione dell'agenda;
2. riproduzione musica;
3. navigazione verso punti di interesse;
4. richiesta di notizie personalizzate;
5. ricerca dell'officina o concessionaria Seat più vicina.

Per *avviare il riconoscimento* vocale è stato integrato sul volante un pulsante, il quale, una volta premuto, mette in ascolto l'assistente. Tutte le feature introdotte con questa novità sui veicoli Seat sono sicuramente utili, ma l'assistente che si ha intenzione di presentare con il nostro progetto ha obiettivi diversi e in qualche modo più ambiziosi. Qui di seguito elencate alcune differenze che esistono tra l'assistente che è stato sviluppato con questa tesi e la proposta di Seat:

- l'assistente *manca* di integrazione con il veicolo, in altre parole manca il monitoraggio di ciò che sta accadendo a livello di motore, ma anche semplicemente il controllo di fanali, tergicristalli o quant'altro;
- cosa più importante, l'assistente è un assistente **reattivo**, quindi la conversazione è sempre avviata dal guidatore e, in questo caso, Alexa si limita semplicemente a rispondere alle richieste/comandi dell'utente.

2.2 Mercedes e “Hey Mercedes”



Figura 2.3: Visione lato guidatore di un cruscotto Mercedes [7]



Figura 2.4: Visione lato guidatore di un cruscotto Mercedes [8]

Anche Mercedes ha presentato la sua idea di assistente alla guida al CES di Las Vegas del 2018. Sicuramente un passo avanti rispetto al progetto Seat, infatti

è un assistente *interamente* prodotto in casa Mercedes [9]. Questa nuova interfaccia guidatore-veicolo viene suddivisa in 3 livelli [7]. Il primo livello è quello dell'**Homescreen** che ha il compito di mostrare tutte le funzionalità principali insieme alle informazioni più importanti. Il secondo livello, il **Basescreen**, mostra il display assieme ai controlli per tutte le applicazioni principali, le funzioni accessibili in questo livello vengono presentate in fondo allo schermo. Il terzo ed ultimo livello è un sottomenù nel quale risiedono le impostazioni raramente utilizzate dall'utente. Di seguito ciò che può fare questo assistente:

1. inserimento di un proprio profilo preferito: regolazione sedili, luci d'ambiente a LED, stazione radio preferita, orientamento della mappa per la navigazione;
2. navigazione;
3. interazione tramite linguaggio colloquiale con l'assistente, infatti, per esempio, pronunciando la frase “hey Mercedes, ho caldo”, l'assistente capirà di dovere abbassare la temperatura nell'abitacolo;
4. calcolo di percorsi dinamicamente a seconda della situazione di traffico (protocollo di comunicazione **Car-to-X** [9]);
5. **In-Car-Office** [9] è una funzionalità che permette l'accesso, direttamente a bordo della macchina, a determinate funzioni come, ad esempio, lettura di appuntamenti, chiamate, ...

In questo caso, diversamente da quello di Seat, per avviare il riconoscimento vocale è necessario pronunciare il comando “Hey Mercedes”, Mercedes-Benz User eXperience (MBUX) allora sarà in grado di capire che si vuole impartire un comando e allora si metterà in ascolto per una richiesta. Anche in questo caso, però, ci sono delle differenze con quello che è stato sviluppato in questa tesi:

- l'assistente di Mercedes **distrae** l'utente dalla guida, infatti per accedere a molte funzionalità, il guidatore, è costretto a ricorrere all'utilizzo del touchscreen, inoltre sono provvisti, per la selezione di alcune impostazioni, dei pulsanti touch sul volante;
- anche in questo caso l'integrazione con il veicolo è ridotta, ci si limita a potere selezionare il canale radio, utilizzare il navigatore, ma nulla riguardante l'effettivo stato del veicolo;
- come per l'assistente Seat, anche quello Mercedes, è un assistente **reattivo**, infatti è sempre l'utente che deve iniziare la conversazione.

2.3 BMW e “Hey BMW”

Una citazione va fatta anche a BMW che ha annunciato che a Marzo 2019 presenterà il suo personale assistente alla guida “Hey BMW” [10], un assistente in grado di migliorarsi comando dopo comando grazie all’intelligenza artificiale [11]. Per quanto annunciato non aggiunge molto di più a quello presentato da Mercedes, tranne per alcune feature:

1. possibilità di sovrascrivere il comando per attivare l’assistente con un nome a piacere, ad esempio “Hey Luca”;
2. possibilità di chiedere istruzioni, per esempio come potere fissare il seggiolino per bambini;
3. integrazione con Microsoft Office 365;
4. compatibilità con altri assistenti vocali, come, ad esempio, Amazon Alexa.

2.4 Analisi del confronto

In conclusione ciò che è presente al momento sul mercato sono soluzioni per assistenti alla guida che però non si comportano da veri e propri assistenti. L’idea fondamentale in questa tesi è che un assistente non debba limitarsi a migliorare l’esperienza di guida o a portare a termine le richieste del guidatore. Sicuramente quelle sono caratteristiche necessarie, ma per essere veramente di aiuto all’utente finale, un assistente ha bisogno di sapersi sostituire all’utente in quei compiti che un guidatore, da solo, non sarebbe in grado di portare a termine. Riferendosi all’esempio di un guasto al motore citato nel capitolo introduttivo, un guidatore non pratico dei segnali sul cruscotto del veicolo, non può essere a conoscenza del fatto che particolari combinazioni di spie possano significare danni gravi al motore, proprio per questo l’assistente deve colmare questa mancanza e avvisarlo di tale eventualità.

Non è solo quello della **reattività** il problema che si intende fare trasparire da questo confronto, ma anche quello dell’invasività degli assistenti di BMW e Mercedes in particolare. Infatti il guidatore si trova sommerso da molteplici touchscreen, leve, bottoni per l’attivazione delle diverse funzionalità dell’assistente e ciò risulta molto distraente, spostando l’attenzione dell’utente via dalla strada. Per questo motivo il lavoro in questa tesi è stato focalizzato allo sviluppo di un assistente che proponga interazioni con il guidatore molto semplici e poco distraenti: quella vocale e quella visiva, senza bisogno di cercare sottomenù in schermi per attivarlo.

Capitolo 3

Casi d'uso

Nel capitolo precedente è stato presentato un confronto con le soluzioni di alcune case automobilistiche attualmente in commercio. Analizzato quindi il contesto nel quale si vuole inserire questo progetto, la fase successiva di questa tesi è stata quella di definire una serie di casi d'uso con i quali testare le effettive capacità dell'assistente sviluppato.

Il lavoro descritto in questo capitolo sarà suddiviso nella seguente maniera: una sezione sarà dedicata alla descrizione delle proprietà chiave dell'assistente 3.1; un'altra sezione si occuperà di definire le categorie di casi d'uso che sono state considerate e del perchè di tale scelta 3.2; infine un'ultima parte descriverà gli effettivi casi d'uso sui quali l'assistente è stato testato 3.3.

3.1 Idea generale



Figura 3.1: Diagramma che descrive come avviene l'interazione con l'assistente

L'idea generale, come anticipato nel capitolo introduttivo 1, è quella di un applicativo che sia in grado di esaudire le richieste dell'utilizzatore, mantenendo sempre sotto controllo ciò che accade all'interno del veicolo e che sia in grado di notare situazioni critiche e reagire di conseguenza. Nella Figura 3.1 viene espresso sottoforma di disegno il ruolo dell'assistente nella comunicazione utente-veicolo:

1. nella comunicazione guidatore-assistente il suo ruolo sarà quello di comprendere la richiesta e portarla a termine;
2. nella comunicazione assistente-guidatore, invece, l'applicativo dovrà costantemente monitorare e quindi essere in comunicazione con il veicolo, di modo da potere notificare l'utente in caso di situazioni particolari.

Si può notare, già da subito, come l'interazione con tale intelligenza sia **bidirezionale** o non unidirezionale come per i casi presentati nel capitolo precedente 2, dove gli assistenti si "limitavano" semplicemente a portare a termine le richieste ricevute dal guidatore. Tale tipo di comunicazione e il fatto che si sia in costante comunicazione con il veicolo monitorando i dati al fine di individuare situazioni particolari, conferiscono a questo tipo di assistente la capacità di essere **proattivo** e non più semplicemente **reattivo** come dimostrato per gli altri casi (sempre facendo riferimento alla sezione 2).

3.2 Categorie di casi d'uso considerati

Date le proprietà necessarie dell'assistente, sono state definite alcune categorie di casi d'uso che coprissero tutte quelle situazioni in cui l'assistente avrebbe potuto esprimere le sue reali potenzialità. Alcune di queste categorie sono state utilizzate per testare capacità già presenti negli assistenti presenti nel commercio, altre sono state definite per testare l'effettiva proprietà proattiva che distingue questo assistente dagli altri. Di conseguenza lo sviluppo è stato focalizzato alla gestione solo di un sottoinsieme di situazioni che si possono verificare durante la guida.

Di seguito verranno descritte le categorie di casi d'uso che sono state prese in considerazione. Le due principali macro-sezioni sono distinguibili in base a chi inizia la conversazione. A loro volta esse contengono categorie più specifiche:

- **Guidatore→Auto**: contengono al loro interno tutte quelle situazioni in cui è il guidatore a interpellare l'assistente. In questo gruppo è possibile identificare due ulteriori sotto-categorie:
 1. Comando
 2. Richiesta di informazioni
- **Auto→Guidatore**: al contrario della precedente, questa macro categoria contiene al suo interno tutte quelle situazioni in cui è l'assistente ad iniziare la conversazione. Anche in questo caso sono stati identificati dei sotto-gruppi:

1. Avvisi e suggerimenti
2. Alert critici

3.2.1 Da guidatore: Comando

Come anticipato nella sezione precedente 3.2, questa categoria fa parte di quel gruppo di casi d'uso dove la conversazione viene iniziata dal guidatore.

Una volta ricevuto il comando, il veicolo, dovrà processare la richiesta e produrre una risposta dipendente dalla fattibilità dell'azione da portare a termine:

1. se l'assistente è in grado di portare a termine l'operazione **automaticamente** allora lo farà;
2. se l'azione non è **automatica** allora l'assistente guiderà l'utente nell'esecuzione manuale.

È evidente, quindi, che l'assistente dovrà possedere le informazioni a proposito dei possibili comandi che è in grado di processare, ma, cosa più importante, dovrà avere conoscenza di ciò che può essere processato in maniera automatica e cosa no e nel caso non possa eseguire lui da solo l'operazione deve possedere le conoscenze necessarie per potere guidare l'utente nell'attuazione dell'azione richiesta.

3.2.2 Da guidatore: Richiesta informazioni

Anche questa categoria, come la precedente, fa parte di quella serie di casi d'uso dove la conversazione è iniziata dall'utente.

Similmente a ciò che è stato detto per il tipo precedente, anche qui il veicolo dovrà essere in grado di processare la richiesta e rispondere adeguatamente, fornendo le informazioni richieste dall'utente.

L'applicazione, in caso di richieste che abbiano bisogno di più specificità da parte del guidatore, dovrà strutturare la conversazione in maniera tale da potere ottenere tutte le informazioni necessarie per fornire all'utente la risposta adeguata.

L'assistente perciò avrà bisogno di avere informazioni a proposito delle funzioni e degli accessori disponibili in auto, con relative varie posizioni di eventuali leve/bottoni.

3.2.3 Da veicolo: Avvisi

Questa categoria di casi d'uso, al contrario delle precedenti due, fa parte della sezione di casi d'uso in cui la conversazione viene avviata dal veicolo.

In particolare, gli avvisi consistono in notifiche, da parte dell'auto di eventuali situazioni anomale, ma **non critiche**. Di conseguenza, l'assistente dovrà essere in grado

di capire quando, il verificarsi di alcune condizioni, può consistere in un'anomalia e di seguito riuscire ad avvisare il guidatore.

Essendo che queste situazioni non sono di natura critica, l'assistente non dovrà essere troppo insistente e distrarre l'utente dalla guida.

3.2.4 Da veicolo: Alert critici

Anche quest'ultima categoria fa sempre parte di quel gruppo di casi d'uso in cui la conversazione viene iniziata dal veicolo, ma in questo caso, differentemente dal paragrafo precedente, il veicolo notificherà il guidatore di una situazione anomala e **critica**.

L'assistente dovrà quindi essere in grado di gestire eventuali richieste di informazioni e/o azioni che il guidatore potrà porre per capire come risolvere tale situazione.

L'applicazione avrà, quindi, bisogno di sapere quali sono i fattori che caratterizzano una situazione critica e sapere fornire informazioni all'utente su come comportarsi.

3.2.5 Motivazione della scelta

La scelta di esattamente queste categorie di casi d'uso è stata fatta per due ragioni principali:

1. la prima è quella di mantenere una certa continuità con gli assistenti presenti in commercio, ma modificandone gli obiettivi. Infatti essi sono già in grado di soddisfare richieste dell'utente, ma tali richieste non riguardano spesso il veicolo stesso. Per tale motivo si è deciso di spostare l'attenzione da comandi come "Imposta tale destinazione sul navigatore" a comandi più inerenti al veicolo in sé, come ad esempio "Accendi i fendinebbia". Lo stesso vale per la richiesta di informazioni su accessori a bordo del veicolo;
2. la seconda è quella di dimostrare le proprietà in più presenti su questo assistente rispetto a quelli presentati nel capitolo 2. Infatti, al fine di testare l'effettiva proattività dell'assistente, sono stati definiti dei casi d'uso che testassero sia la capacità dell'applicativo di monitorare il veicolo e identificare situazioni anomale, sia la sua capacità di essere il primo ad iniziare una conversazione con il guidatore.

3.3 Casi d'uso implementabili

Come anticipato nella sezione 3.2 le situazioni che l'applicativo è in grado di affrontare sono un sottoinsieme di tutte le situazioni reali. In questa sezione verranno affrontati e descritti i **casi d'uso** che l'assistente conversazionale può gestire e

rispetto ai quali è stato successivamente testato il suo effettivo funzionamento. Ogni sotto sezione è stata suddivisa in diverse parti:

- *Descrizione del caso d’uso*: viene presentata la situazione nella quale può trovarsi il guidatore;
- *Variazioni possibili alla conversazione*: vengono espone le possibili variazioni della conversazione, presentata precedentemente, l’assistente è in grado di capire;
- *Aspetti del veicolo interessati*: vengono spiegati gli aspetti del veicolo dei quali l’assistente ha bisogno per portare a termine la richiesta dell’utente;
- *Azioni compiute dall’intelligenza del veicolo*: vengono spiegate, passaggio per passaggio, tutte le operazioni che l’applicazione deve compiere per portare a termine ciò che le è stato richiesto;
- *Caratteristiche della conversazione*: viene spiegato come è possibile che la conversazione avvenga e come l’assistente sia in grado di capire ciò che è stato richiesto.

3.3.1 Da guidatore: Comando

Questo caso d’uso fa parte del gruppo che si riferisce all’interazione utente-veicolo, di conseguenza la conversazione viene iniziata dall’utente. Il veicolo avrà bisogno, quindi, di processare le richieste ricevute dal guidatore di modo da riuscire ad ottenere dati relativi al contesto della conversazione e rispondere (o effettuare azioni) propriamente, basandosi sulle informazioni contenute all’interno del database.

Descrizione del caso d’uso

Contesto: Si sta guidando in condizioni di poca visibilità (nebbia).

Guidatore: “Accendi i fendinebbia”

Veicolo: “Quali?”

Guidatore: “Anteriori”

Veicolo: “Azione effettuata”

Variazioni possibili alla conversazione

L’assistente è in grado di riconoscere anche altre frasi di input per iniziare il processamento delle informazioni per questo caso d’uso. Ad esempio:

- *“Come si accendono i fendinebbia”*
In questo caso l'assistente esporrà all'utente le azioni necessario per l'accensione dei fanali offrendosi anche di effettuare lui l'azione.
- *“Aziona i fendinebbia”, “Attiva i fendinebbia”*
L'assistente capirà che l'utente ha richiesto l'accensione dei fendinebbia.
- *“Spegni i fendinebbia”, “Disattiva i fendinebbia”*
L'assistente capirà che l'azione richiesta dall'utente è l' spegnimento dei fendinebbia.

Lo stesso stile di conversazione può essere utilizzato per tutti gli altri accessori che l'assistente possiede nel database ed è in grado di riconoscere.

Aspetti del veicolo interessati

Per questo caso d'uso l'intelligenza del veicolo avrà bisogno di alcune informazioni sul veicolo:

- quali accessori che è possibile accendere/spegnere, sono disponibili sul veicolo;
- istruzioni per informare l'utente su come effettuare l'azione richiesta;
- come portare a termine l'azione richiesta.

Azioni compiute dall'intelligenza del veicolo

L'assistente performerà alcuni step prima di potere fornire un responso al guidatore:

1. ricezione richiesta dell'utente tramite comando vocale;
2. invio del payload audio al cloud-service (in questo caso Google Speech-to-Text) in modo da potere ottenere una trascrizione di ciò che è stato richiesto dall'utente;
3. invio della trascrizione ad un altro cloud-service (in questo caso DialogFlow) in grado di potere derivare, da una frase, il contesto della conversazione e le informazioni necessarie all'adempimento delle azioni richieste;
4. elaborazione, in base alle informazioni estrapolate dal secondo cloud service, di un responso adeguato da ritornare all'utente;
5. invio del responso ad un terzo cloud-service (in questo caso Google Text-to-Speech) in grado di ritornare, dato un testo, l'audio di tale frase;
6. scrittura del file audio ritornato dal terzo cloud-service;

7. riproduzione del file audio;

8. cancellazione del file audio.

Caratteristiche della conversazione

La conversazione sarà sviluppata tramite riconoscimento di parole chiave. Ciò significa che è necessario “allenare” l’intelligenza dell’assistente a riconoscere parole come “fendinebbia” e associarle al relativo accessorio salvato all’interno del database locale. Stesso ragionamento è da applicare alle parole chiave indicanti azioni (es. accensione, spegnimento, ...).

3.3.2 Da guidatore: Richiesta informazioni

Questo caso d’uso fa parte del gruppo che si riferisce all’interazione utente-veicolo, di conseguenza la conversazione viene iniziata dall’utente. Il veicolo avrà bisogno, quindi, di processare le richieste ricevute dal guidatore di modo da riuscire ad ottenere dati relativi al contesto della conversazione e rispondere (o effettuare azioni) propriamente, basandosi sulle informazioni contenute all’interno del database.

Descrizione del caso d’uso

Contesto: Situazione di guida normale.

Guidatore: “Ho notato una spia accesa sul cruscotto e non so cosa significa”

Veicolo: “Quale tra queste che risultano accese? [...]” (L’assistente espone una lista di spie attualmente accese sul cruscotto dell’auto)

Guidatore: “La seconda”

Veicolo: “Questa spia significa [...]” (L’assistente spiega il significato della spia indicata dal guidatore)

Variazioni possibili alla conversazione

L’assistente è in grado di riconoscere anche altre frasi di input per iniziare il processamento delle informazioni per questo caso d’uso. Ad esempio:

- “C’è una spia accesa e non so che cosa significa”, “Ho visto una spia accesa e non ne conosco il significato”
- “Quella” (nel caso sia presentata una sola spia), “La prima”, “La terza”, ...
L’assistente capirà che la posizione nella lista della spia indicata dal guidatore.

La conversazione è indirizzata dal veicolo, al fine di semplificare il riconoscimento della spia interessata in caso di più spie con stesso colore e simil forma. Si passerà, di conseguenza, sempre attraverso lo step che prevede l'assistente esporre la lista delle spie attualmente accese sullo schermo. Gli indicatori presentati, ovviamente, sono quelli di cui si conosce il significato e quindi salvati all'interno del database.

Aspetti del veicolo interessati

Per questo caso d'uso l'intelligenza del veicolo avrà bisogno di alcune informazioni sul veicolo:

- spie disponibili con relative informazioni;
- spie attualmente accese sul cruscotto.

Azioni compiute dall'intelligenza del veicolo

L'assistente performerà alcuni step prima di potere fornire un responso al guidatore:

1. ricezione richiesta dell'utente tramite comando vocale;
2. invio del payload audio al cloud-service (in questo caso Google Speech-to-Text) in modo da potere ottenere una trascrizione di ciò che è stato richiesto dall'utente;
3. invio della trascrizione ad un altro cloud-service (in questo caso DialogFlow) in grado di potere derivare, da una frase, il contesto della conversazione e le informazioni necessarie all'adempimento delle azioni richieste;
4. elaborazione, in base alle informazioni estrapolate dal secondo cloud service, di un responso adeguato da ritornare all'utente;
5. invio del responso ad un terzo cloud-service (in questo caso Google Text-to-Speech) in grado di ritornare, dato un testo, l'audio di tale frase;
6. scrittura del file audio ritornato dal terzo cloud-service;
7. riproduzione del file audio;
8. cancellazione del file audio.

Caratteristiche della conversazione

La conversazione sarà sviluppata tramite riconoscimento di parole chiave. Ciò significa che è necessario “allenare” l’intelligenza dell’assistente a riconoscere parole come “spie”, “significa” per triggerare l’inizio della conversazione e in seguito riconoscere parole come “prima”, “seconda”, “quella” in modo da potere ottenere la posizione dell’indicatore interessato all’interno della lista presentata.

3.3.3 Da guidatore: Richiesta di informazioni (derivate)

Questo caso d’uso fa parte del gruppo che si riferisce all’interazione utente-veicolo, di conseguenza la conversazione viene iniziata dall’utente. Il veicolo avrà bisogno, quindi, di processare le richieste ricevute dal guidatore di modo da riuscire ad ottenere dati relativi al contesto della conversazione e rispondere (o effettuare azioni) propriamente, basandosi sulle informazioni contenute all’interno del database.

Descrizione del caso d’uso

Contesto: Situazione di guida dove si è quasi in riserva.

Guidatore: “Ho notato che sono quasi in riserva. Quanti chilometri di autonomia mi rimangono?”

Veicolo: “Vuoi che calcoli considerando la velocità attuale?”

Guidatore: “Sì”

Veicolo: “Ti rimangono 15 chilometri di autonomia”

Variazioni possibili alla conversazione

L’assistente è in grado di riconoscere anche altre frasi di input per iniziare il processamento delle informazioni per questo caso d’uso. Ad esempio:

- “*Quanti chilometri di autonomia mi rimangono*”, “*Chilometri di autonomia*”, “*Autonomia residua*”
- “*Esatto*”, “*Esattamente*”

L’assistente capirà che l’utente data una risposta affermativa.

I chilometri di autonomia saranno sempre calcolati utilizzando la velocità corrente del veicolo.

Aspetti del veicolo interessati

Per questo caso d'uso l'intelligenza del veicolo avrà bisogno di alcune informazioni sul veicolo:

- velocità attuale del veicolo;
- livello di carburante residuo.

Azioni compiute dall'intelligenza del veicolo

L'assistente performerà alcuni step prima di potere fornire un responso al guidatore:

1. ricezione richiesta dell'utente tramite comando vocale;
2. invio del payload audio al cloud-service (in questo caso Google Speech-to-Text) in modo da potere ottenere una trascrizione di ciò che è stato richiesto dall'utente;
3. invio della trascrizione ad un altro cloud-service (in questo caso DialogFlow) in grado di potere derivare, da una frase, il contesto della conversazione e le informazioni necessarie all'adempimento delle azioni richieste;
4. elaborazione, in base alle informazioni estrapolate dal secondo cloud service, di un responso adeguato da ritornare all'utente;
5. invio del responso ad un terzo cloud-service (in questo caso Google Text-to-Speech) in grado di ritornare, dato un testo, l'audio di tale frase;
6. scrittura del file audio ritornato dal terzo cloud-service;
7. riproduzione del file audio;
8. cancellazione del file audio.

Caratteristiche della conversazione

La conversazione sarà sviluppata tramite riconoscimento di parole chiave. Ciò significa che è necessario “allenare” l'intelligenza dell'assistente a riconoscere parole come “velocità”, “chilometri rimasti”, “autonomia”, “sì”, “esatto”, ...

3.3.4 Da auto: Avvisi

Questo caso d'uso fa parte del gruppo che si riferisce all'interazione veicolo-utente, di conseguenza la conversazione viene iniziata dal veicolo. Quest'ultimo avrà bisogno, quindi, di processare le risposte ricevute dal guidatore di modo da riuscire ad ottenere dati relativi al contesto della conversazione e proseguire la conversazione (o effettuare azioni) propriamente, basandosi sulle informazioni contenute all'interno del database.

Descrizione del caso d'uso

Contesto: Situazione di guida normale.

Veicolo: “La luce pressione degli pneumatici è accesa”

Guidatore: “Che cosa significa?”

Veicolo: “Questa spia viene accesa quanto la pressione degli pneumatici è troppo bassa oppure se sono state cambiate le gomme di recente. Hai cambiato le gomme di recente?”

Guidatore: “Sì”

Veicolo: “Perfetto, allora spengo la spia”

Variazioni possibili alla conversazione

L'assistente è in grado di attivarsi nel caso di accensione di un qualsiasi indicatore del suo database. Per il proseguimento della conversazione è possibile riconoscere anche altre frasi dell'utente, ad esempio:

- “*Cosa significa?*”, “*Qual è il suo significato?*”, “*Cosa vuol dire?*”, “*Che devo fare?*”

L'assistente capirà che l'utente ha richiesto maggiori informazioni sulla spia e sulle azioni da fare per potere risolvere un eventuale problema

- “*Esatto*”, “*Esattamente*”

L'assistente capirà che l'utente data una risposta affermativa.

- “*No*”, “*Negativo*”

L'assistente capirà che l'utente data una risposta negativa.

Nell'esempio, nel caso in cui venga data una risposta negativa alla domanda, la spia non cambierà il suo stato attuale.

Aspetti del veicolo interessati

Per questo caso d'uso l'intelligenza del veicolo avrà bisogno di alcune informazioni sul veicolo:

- spie disponibili con relative informazioni;
- notifica di cambiamento dello stato delle spie sul cruscotto.

Azioni compiute dall'intelligenza del veicolo

L'assistente performerà alcuni step prima di potere fornire un responso al guidatore:

1. azionare un evento sul cloud-service (in questo caso DialogFlow) in modo da creare il nuovo contesto di conversazione;
2. ricezione della risposta dell'utente tramite comando vocale;
3. invio del payload audio al cloud-service (in questo caso Google Speech-to-Text) in modo da potere ottenere una trascrizione di ciò che è stato richiesto dall'utente;
4. invio della trascrizione ad un altro cloud-service (in questo caso DialogFlow) in grado di potere derivare, da una frase, il contesto della conversazione e le informazioni necessarie all'adempimento delle azioni richieste;
5. elaborazione, in base alle informazioni estrapolate dal secondo cloud service, di un responso adeguato da ritornare all'utente;
6. invio del responso ad un terzo cloud-service (in questo caso Google Text-to-Speech) in grado di ritornare, dato un testo, l'audio di tale frase;
7. scrittura del file audio ritornato dal terzo cloud-service;
8. riproduzione del file audio;
9. cancellazione del file audio.

Caratteristiche della conversazione

La conversazione sarà sviluppata tramite, in primis, l'attivazione di un evento sul cloud-service e in seguito attraverso il riconoscimento di parole chiave. Ciò significa che è necessario "allenare" l'intelligenza dell'assistente a riconoscere parole come "Significa", "Significato", "Fare", "Sì", "Esatto", "Esattamente", "No".

3.3.5 Da auto: Alert critici

Questo caso d’uso fa parte del gruppo che si riferisce all’interazione veicolo-utente, di conseguenza la conversazione viene iniziata dal veicolo. Quest’ultimo avrà bisogno, quindi, di processare le risposte ricevute dal guidatore di modo da riuscire ad ottenere dati relativi al contesto della conversazione e proseguire la conversazione (o effettuare azioni) propriamente, basandosi sulle informazioni contenute all’interno del database.

Descrizione del caso d’uso

Contesto: Situazione di guida normale.

Veicolo: “La luce del freno a mano e dell’ABS risultano accese”

Guidatore: “Che cosa significa?”

Veicolo: “Indica che ci può essere un problema con il motore. Recati al più presto da un meccanico.”

Guidatore: “Sì”

Variazioni possibili alla conversazione

L’assistente è in grado di attivarsi nel caso di accensione di un qualsiasi indicatore del suo database, ma in questo caso, la conversazione viene iniziata solo nel caso si realizzi una **condizione critica**. Per il proseguimento della conversazione è possibile riconoscere anche altre frasi dell’utente, ad esempio:

- “Cosa significa?”, “Qual è il suo significato?”, “Cosa vuol dire?”, “Che devo fare?”

L’assistente capirà che l’utente ha richiesto maggiori informazioni sulla spia e sulle azioni da fare per potere risolvere un eventuale problema

Essendo questo caso d’uso attivato solamente in una situazione critica, le spie non cambieranno il loro stato.

Aspetti del veicolo interessati

Per questo caso d’uso l’intelligenza del veicolo avrà bisogno di alcune informazioni sul veicolo:

- spie disponibili con relative informazioni;
- notifica di cambiamento dello stato delle spie sul cruscotto;
- informazioni per riconoscere quando una situazione risulta critica.

Azioni compiute dall'intelligenza del veicolo

L'assistente performerà alcuni step prima di potere fornire un responso al guidatore:

1. azionare un evento sul cloud-service (in questo caso DialogFlow) in modo da creare il nuovo contesto di conversazione;
2. ricezione della risposta dell'utente tramite comando vocale;
3. invio del payload audio al cloud-service (in questo caso Google Speech-to-Text) in modo da potere ottenere una trascrizione di ciò che è stato richiesto dall'utente;
4. invio della trascrizione ad un altro cloud-service (in questo caso DialogFlow) in grado di potere derivare, da una frase, il contesto della conversazione e le informazioni necessarie all'adempimento delle azioni richieste;
5. elaborazione, in base alle informazioni estrapolate dal secondo cloud service, di un responso adeguato da ritornare all'utente;
6. invio del responso ad un terzo cloud-service (in questo caso Google Text-to-Speech) in grado di ritornare, dato un testo, l'audio di tale frase;
7. scrittura del file audio ritornato dal terzo cloud-service;
8. riproduzione del file audio;
9. cancellazione del file audio.

Caratteristiche della conversazione

La conversazione sarà sviluppata tramite, in primis, l'attivazione di un evento sul cloud-service e in seguito attraverso il riconoscimento di parole chiave. Ciò significa che è necessario "allenare" l'intelligenza dell'assistente a riconoscere parole come "Significa", "Significato", "Fare".

Capitolo 4

Progettazione del sistema

In questo capitolo verranno presentati i componenti del sistema (con i relativi ruoli) e come è stata pensata l'interazione tra di loro.

Importante appunto che va fatto è quello che tutte le scelte effettuate sono state fatte tenendo a mente che l'applicazione è pensata per essere utilizzata su centraline con capacità di calcolo limitate (Raspberry PI ¹). Inoltre, questa versione del prototipo supporta solamente la lingua italiana.

4.1 Architettura del sistema

Il contesto nel quale va ad inserirsi il prototipo è di veicoli connessi, questo ha consentito di potere sfruttare servizi cloud per lo svolgimento di alcuni compiti. Il risultato ottenuto è quello di un'applicazione che risulta molto modularizzata e ciò ha due principali vantaggi:

1. il carico computazionale all'interno del veicolo è enormemente ridotto e questo facilita l'utilizzo dell'applicazione su centraline con poche risorse, come anticipato precedentemente;
2. l'aggiornamento/sostituzione dei singoli componenti risulta molto semplice.

Nella Figura 4.1 è possibile notare come tali servizi cloud citati in precedenza siano stati sfruttati come Conversational Engine (ossia quel modulo incaricato di strutturare la conversazione, riconoscere parole chiave e intenti delle richieste dell'utente) e come Speech Synthesis & Voice Recognition (moduli necessari alla traduzione da input vocale a input testuale e viceversa).

Sempre in riferimento alla Figura 4.1, l'interazione con il prototipo è limitata al modulo di Car & HMI Simulator, responsabile di fornire all'utente finale un ambiente

¹<https://www.raspberrypi.org/>. *Ultimo accesso:* 07/03/2019

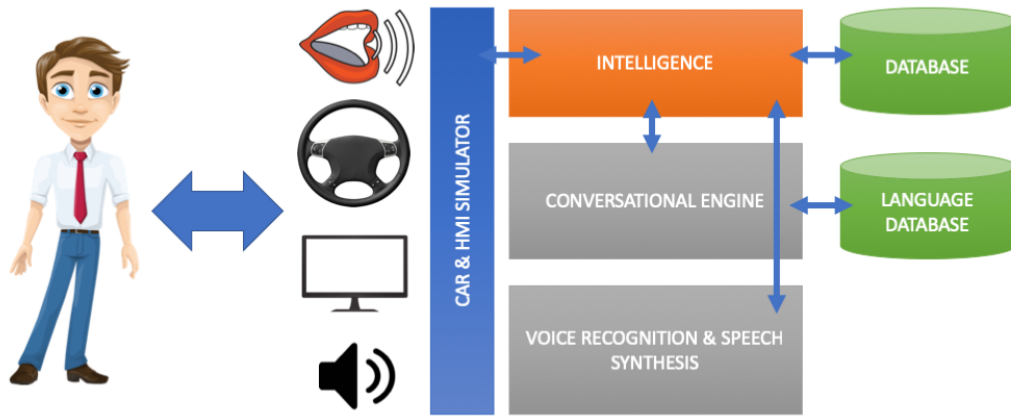


Figura 4.1: Architettura del sistema e interazione fra i componenti.

di guida il più simile possibile a quello di un veicolo reale. Inoltre questo modulo è anche responsabile dell'interazione vocale con l'assistente e di fornire dati riguardanti lo stato del veicolo all'intelligenza del sistema.

4.1.1 Voice Recognition & Speech Synthesis

Questo modulo ha il ruolo di gestire l'interazione vera e propria con il guidatore. È responsabile di ricevere l'input vocale dall'utente e trasformarlo in testo per l'elaborazione, ma anche del contrario, ossia trasformare il responso creato dall'intelligenza del sistema (in forma testuale) in responso vocale. Inoltre questo modulo dell'applicativo, sarà quello che si occuperà della preparazione dei dati per il cloud-service designato per il Natural Language Processing (NLP), necessario al fine di ottenere contesto della conversazione, oggetti di interesse e altre informazioni deducibili dalla richiesta dell'utente per poter produrre un risultato corretto.

Scelta dei componenti per Speech Synthesis

Come detto in precedenza, essendo la capacità della centralina limitata e trovandoci in un contesto di connected vehicles, si è deciso di utilizzare un cloud service per la conversione *text-to-speech* e *speech-to-text*. In particolare i servizi cloud utilizzati sono stati quelli messi a disposizione da Google: Google Cloud Speech-to-Text ² e

²<https://cloud.google.com/speech-to-text/>. *Ultimo accesso: 03/03/2019*

Google Cloud Text-to-Speech ³. Il primo è stato utilizzato per la conversione da input vocale del guidatore a formato testuale, più semplice per l'elaborazione e il secondo per trasformare il risultato prodotto dall'intelligenza dell'applicazione in risposta vocale da presentare al guidatore.

Relazioni con altri moduli

Questo modulo è aperto su un unico fronte principale, quello con l'applicazione sul veicolo contenente l'intelligenza del sistema. Indirettamente si può affermare che tra questo modulo e l'utente vi sia un'interazione vocale, ma vi è sempre il passaggio attraverso l'applicazione principale.

L'utente interagisce con l'assistente attraverso il microfono, le sue richieste vengono catturate ed inviate ai cloud services del modulo per la Voice Recognition e Speech Synthesis. Questi ultimi si preoccupano di tradurre in forma testuale l'input vocale dell'assistente, di modo che sia già nella forma desiderata dal modulo per la Conversational Engine o viceversa se si deve presentare una risposta al guidatore. Quindi riferendosi alla Figura 4.1, i dati che viaggiano sul canale di comunicazione tra intelligenza del sistema e il modulo Voice Recognition & Speech Synthesis, sono di due tipi, dipendenti dalla direzione della comunicazione e dal punto della conversazione in cui ci si trova.

Il formato dei dati che passano tra i due moduli quando l'utente effettua una richiesta è il seguente: nella direzione *Intelligenza ↔ Modulo Voice Recognition & Speech Synthesis* i dati sono vocali, catturati dall'applicazione a bordo del veicolo attraverso il microfono, mentre se si considera la direzione inversa risultano in formato testuale e corrispondono alla trascrizione della richiesta dell'utente.

Quando invece è l'assistente a fornire una risposta all'utente, nella direzione *Intelligenza ↔ Modulo Voice Recognition & Speech Synthesis* i dati sono in formato testuale, sono la risposta elaborata dall'intelligenza del veicolo in collaborazione con il modulo per la Conversational Engine, mentre nella direzione inversa sono in formato vocale e corrispondono alla traduzione della risposta che l'assistente ha deciso di presentare alla richiesta dell'utente.

4.1.2 Conversational Engine & Intelligence

Questi due moduli insieme costituiscono il cuore pulsante dell'applicazione. Qui avviene tutta l'elaborazione dei dati necessaria al fine di produrre il responso corretto in base alle richieste dell'utente. Inoltre l'intelligenza dell'applicazione tiene traccia dello stato degli oggetti all'interno del veicolo e lo aggiorna in base a ciò che il guidatore chiede. Per raggiungere questo obiettivo è stato necessario introdurre una base dati, la quale non ha semplicemente la funzione di memorizzare lo stato

³<https://cloud.google.com/text-to-speech/>. *Ultimo accesso:* 03/03/2019

dell'oggetto in questione, ma si deve preoccupare di mantenere anche la relazione tra spia e relativo oggetto, insieme ad altre informazioni come, ad esempio, la sua posizione nel veicolo o le istruzioni per l'accensione.

Scelta dei componenti

Per la realizzazione di questo modulo è stata necessaria la scelta di due componenti, uno per la *Conversational Engine*, ossia quella parte che si occupa del riconoscimento del contesto della conversazione, oggetti di interesse e altri dati deducibili dalla richiesta del guidatore; un altro invece per la memorizzazione dei dati sul veicolo e l'aggiornamento del loro stato.

Nel primo caso la scelta è ricaduta su *DialogFlow*⁴, un servizio cloud offerto da Google utilizzato per la modellazione/gestione di conversazioni.

Nel secondo caso, invece, è stato deciso di utilizzare *SQLite*, un tipo di base dati funzionante⁵ molto bene in ambito embedded.

Confronto assistenti conversazionali

Prima della scelta di *DialogFlow*⁴ come *Conversational Engine* utilizzata dal nostro assistente, sono state analizzate diverse soluzioni in commercio. In particolare sono state presi in considerazione, oltre al framework offerto da Google, anche *Wit.ai*⁶ e *IBM Watson*⁷.

Il confronto fra essi è stato fatto su diversi fronti, come è possibile evincere dalla figura 4.2: lingue supportate, linguaggi di programmazione supportati, pricing, parsing dell'input, qualità della documentazione e facilità di utilizzo.

Per quanto riguarda le *lingue supportate*, tutte e tre le *Conversational Engines* supportano una grande varietà di idiomi, compreso l'italiano che è stata la lingua scelta sulla quale basare l'interazione con l'assistente. *DialogFlow*, però, nella documentazione presenta specifiche più particolari rispetto a *Wit.ai* e *IBM Watson*, precisando il tipo di conversazione più adatta per ciascuna delle lingue supportate.

Un altro aspetto che è stato considerato nel confronto è stato quello del *pricing* per l'utilizzo del prodotto. *DialogFlow* e *Wit.ai* sono entrambi ad utilizzo gratuito, al contrario di *IBM Watson*.

⁴<https://dialogflow.com/>. *Ultimo accesso:* 03/03/2019

⁵<https://www.sqlite.org/index.html>. *Ultimo accesso:* 07/03/2019

⁶<https://wit.ai/>. *Ultimo accesso:* 03/03/2019

⁷<https://www.ibm.com/watson>. *Ultimo accesso:* 03/03/2019

	DialogFlow (Google)	Wit.ai	IBM Watson Assistant (IBM)
Lingue supportate	Supporta tutte le lingue, la lingua inglese è quella più usata e sviluppata. Nella documentazione viene spiegato per quali tipi di conversazione le varie lingue sono più adatte.	Supporta diverse lingue, ma non viene spiegato nella documentazione in quali tipi di conversazione è meglio utilizzare quali lingue.	Sono supportate praticamente tutte le lingue, ma come nel caso di Wit.ai non è spiegato a quali conversazioni si adattano meglio.
Linguaggi di programmazione supportati	Supportati praticamente tutti i maggiori linguaggi di programmazione.	Node.js, Python, Ruby.	Supportati quasi tutti i maggiori linguaggi di programmazione. Meno che DialogFlow.
Pricing	Free to use.	Free to use.	A pagamento, ma utilizzabile con la licenza del Politecnico.
Parsing	Non è presente una chiara spiegazione di come l'assistente effettua il parsing dell'input.	Per il parsing dell'input utilizza Duckling. Nella documentazione è detto che sono presenti piccoli problemi con i riferimenti temporali. Ad esempio, se l'utente dicesse "1 anno dopo Natale", l'assistente farebbe come assunzione che lo user si riferisse al Natale scorso. Per indicare Natale di quest'anno sarebbe necessario esplicitarlo.	Non è presente una chiara spiegazione di come l'assistente effettua il parsing dell'input.
Pareri personali e user experience	Molto valida conversational engine. Interfaccia user friendly, documentazione ben fatta ed è ricca di feature (ad esempio vi è pure una sezione per le analytics delle conversazioni). Fattori in più sono il fatto che sia gratuito e Google è un'etichetta che ha sempre sfornato prodotti di qualità.	Nulla di speciale, la documentazione è un poco scarna. L'interfaccia grafica non è molto immediata come quella di DialogFlow.	È un ottimo prodotto, molto facile capire dove stanno le cose e come funziona il tutto. L'interfaccia grafica è user friendly e molto facile abituarsi. Tuttavia, per lo sviluppo vi è comunque bisogno della licenza o del permesso. IBM comunque è una grande azienda e il prodotto sembra molto solido e affidabile.

Figura 4.2: Risultati del confronto svolto tra tre assistenti conversazionali presenti sul mercato: DialogFlow, Wit.ai, IBM Watson.

Le possibilità di scelta si sono ristrette quando i tre prodotti sono stati confrontati dal punto di vista dei *linguaggi di programmazione* supportati. Partendo dal presupposto che il linguaggio che si è deciso di utilizzare per lo sviluppo dell'applicazione fosse Java, la Conversational Engine prescelta avrebbe dovuto fornire la SDK corrispondente da utilizzare. Wit.ai è l'unica Conversational Engine che si è dimostrata non supportare tale linguaggio, fornendo solamente la possibilità di essere utilizzata in Node.js, Python e Ruby.

Uno degli ultimi aspetti rispetto ai quali sono state confrontate le diverse Conversational Engine è stato quello del *parsing* dell'input. È un aspetto secondario, ma che se conosciuto e spiegato dal framework può facilitare la fase di debugging e la comprensione di come le frasi ricevute in input vengano parsificate. L'unica soluzione che fornisce questo tipo di informazione è Wit.ai che utilizza Duckling⁸. Quest'ultimo è un prodotto che permette la parsificazione del testo e conseguente creazione di dati strutturati. Tale framework, però, presenta ancora delle limitazioni come il mancato supporto per l'italiano.

⁸<https://duckling.wit.ai/>. *Ultimo accesso: 05/03/2019*

Dato questo confronto, la scelta finale è ricaduta su DialogFlow e le motivazioni principali sono state:

- facilità di utilizzo: l'interfaccia grafica della web application per strutturare la conversazione è molto immediata e semplice da utilizzare;
- viene offerta la possibilità di ottenere dati statistici sull'utilizzo dei percorsi nella conversazione;
- è un framework gratuito;
- fornisce una SDK per Java;
- è supportata la lingua italiana.

Relazioni con altri moduli

Conversational Engine e intelligenza del sistema sono in comunicazione tra loro per potere derivare dalle richieste/risposte dell'utente il contesto della conversazione e gli oggetti di interesse. Tra di loro i dati scambiati sono in formato testuale. Inoltre l'intelligenza del sistema è aperta su due ulteriori fronti:

1. con il modulo per Speech Synthesis e Voice Recognition: come detto nel paragrafo precedente 4.1.1, la comunicazione con questo modulo è necessaria per la trascrizione dell'input vocale dell'utente e la trasformazione da testo ad audio delle risposte da presentare al guidatore elaborate dall'assistente;
2. con il modulo Car e HMI Simulator al fine di ottenere sia l'input dell'utente attraverso il microfono, ma soprattutto per ottenere dati del veicolo, visualizzarli a video e visualizzare il risultato delle richieste del guidatore.

4.1.3 Car Simulator & HMI Simulator

Questi sono gli ultimi due moduli del sistema. Nel prototipo sviluppato hanno l'obiettivo di simulare l'hardware e il software preesistenti nel veicolo e con il quali l'applicativo dovrà integrarsi. Si occupano di fornire all'utente un'interfaccia chiara con la quale interagire.

Scelta dei componenti

L'unica scelta che è stata fatta è quella sul simulatore di guida, che però ha portato direttamente ad Assetto Corsa⁹, rinomato videogame/simulatore di guida

⁹<https://www.assettocorsa.net/home-ac/>. *Ultimo accesso: 03/03/2019*

che permette (nell'ambito di determinate gare) di accedere facilmente ai dati del veicolo.

Relazioni con altri moduli

In questo caso il modulo dovrà interagire su un unico fronte, quello verso il modulo di Conversational Engine & Intelligence: dovrà mantenere un flusso costante di dati per aggiornare all'interno dell'applicativo la situazione attuale del veicolo, fornendo, ad esempio, informazioni sulla velocità, ma anche ricevere a sua volta informazioni da mostrare all'utente.

Capitolo 5

Implementazione

In questo capitolo verrà descritto come è stata implementata l'applicazione vera e propria. In particolare prima si presenteranno le scelte implementative effettuate a livello di simulatore e di conversational engine assieme al loro utilizzo, poi verrà spiegato il cuore dell'applicazione e il ruolo delle sue diverse sezioni. Infine verrà esposto come entrano in azione i vari moduli in un esempio di caso d'uso.

La parte di codice che compone la logica interna del prototipo è stata scritta in Java. Inoltre è stata aggiunta una parte di cruscotto integrativa, ai fini della simulazione utilizzando Angular ¹. Come anticipato in precedenza, questo componente è stato aggiunto al fine di compensare quello che il cruscotto del simulatore non offriva, ossia la possibilità di visualizzare eventuali spie di accessori presenti sul veicolo.

5.1 Architettura e comunicazione fra i componenti

Nella Figura 5.1 è possibile vedere come è stata organizzata la comunicazione tra i diversi componenti del sistema. La *user experience* si limita solo all'interazione con alcuni componenti di facciata, in particolare:

- Assetto Corsa: il simulatore di guida utilizzato per ricreare una situazione reale;
- dashboard integrativa: dove sarà possibile vedere alcuni dati estratti da quelli di Assetto Corsa ricevuti dall'applicazione e altri dati aggiunti ad hoc per il prototipo (spie degli accessori che è possibile accendere/spegnere);
- componente per l'interazione vocale: utilizzato per ricevere le richieste del guidatore e presentare le risposte elaborate dall'applicazione.

¹<https://angular.io/>. *Ultimo accesso*: 11/03/2019

La maggior parte dell'applicativo è nascosta all'utente e si occuperà del processamento delle domande del guidatore, interagendo con gli adeguati *cloud services*.

Come si evince dalla Figura 5.1, l'applicazione centrale è stata suddivisa in diversi moduli, discussi in seguito nella sezione 5.4, ognuno responsabile di un particolare compito.

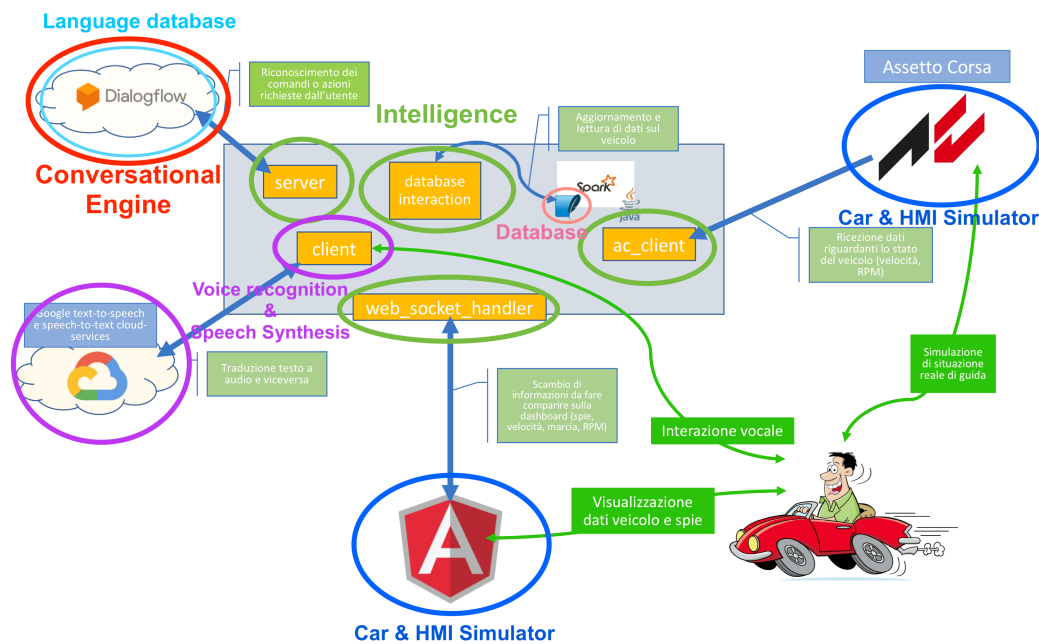


Figura 5.1: Rappresentazione di come avviene la comunicazione tra i vari componenti del prototipo e la loro suddivisione.

La suddivisione presentata nella Figura 4.1 del capitolo precedente viene declinata nell'applicazione reale come mostrato in Figura 5.1. I vari componenti della rappresentazione sono raggruppati come segue:

- *Voice Recognition & Speech Synthesis*: evidenziato in viola nella Figura 5.1. Di questo modulo fanno parte i cloud service offerti da Google per la traduzione text-to-speech/speech-to-text e client (una sezione dell'applicazione principale responsabile dell'interazione vocale con il guidatore). Il loro utilizzo avviene nell'interazione con l'utente per la traduzione in formati opportuni dei dati (vocale o testuale);

- *Conversational Engine*: evidenziato in rosso nella Figura 5.1. Come anticipato in precedenza questo è il modulo responsabile della strutturazione della conversazione e del riconoscimento di parole chiave all'interno delle richieste dell'utente al fine di determinarne gli intenti. Tali compiti sono stati assegnati al cloud service DialogFlow;
- *Car Simulator & HMI Simulator*: evidenziato in blu nella Figura 5.1. A questo modulo corrispondono i componenti Assetto Corsa, dashboard integrativa sviluppata in Angular. Sono utilizzati per l'interazione visuale e di guida con l'utente;
- *Intelligence*: evidenziato in verde nella Figura 5.1. In questo modulo sono contenute tutte quelle parti dell'applicativo centrale responsabili della logica interna (server), dell'interazione con il simulatore per la raccolta dei dati (ac_client), dell'interazione con il cruscotto integrativo per la visualizzazione dei dati (web_socket_handler) e dell'aggiornamento della base dati (database_interaction).

5.2 Interazione utente-simulatore

Nelle Figure 5.2 e 5.3 è rappresentata una visione frontale del simulatore assemblato per questo prototipo, così come apparirebbe a un'utilizzatore.

L'interazione tra guidatore e veicolo avviene principalmente attraverso volante, marce e pedali che simulano l'esperienza di guida. Inoltre l'utente ha la possibilità di vedere il risultato delle richieste da lui fatte all'assistente e di dati del veicolo sul video della dashboard integrativa. Quest'ultima è stata aggiunta in secondo luogo essendo che il cruscotto di Assetto Corsa non era modificabile e conteneva una quantità minima di informazioni sul veicolo.

Nella Figura 5.2 è possibile vedere evidenziato il modulo Intelligence del sistema, tale modulo è il centro dell'interazione tra tutte le parti del sistema e contiene la logica dell'assistente. In particolare, in questo caso, viene utilizzato il microfono di quell'elaboratore, per potere comunicare i comandi o le risposte all'assistente.



Figura 5.2: Figura che evidenzia il modulo contenente l'Intelligence del sistema e Dashboard integrata.



Figura 5.3: Figura che evidenzia i moduli per Car Simulator & HMI Simulator.

Infine nella Figura 5.3 è possibile trovare evidenziato il modulo con funzione di Car Simulator & HMI Simulator. Come anticipato in precedenza questo è il modulo

responsabile di simulare la vera e propria esperienza di guida con l'utente finale. L'interazione tra guidatore ed esso avviene attraverso l'utilizzo di volante, marce e pedali.

5.3 Componenti nel prototipo

5.3.1 Assetto Corsa

Assetto Corsa è stato utilizzato soprattutto per la facilità con cui è possibile reperire informazioni a proposito di dati telemetrici e alla presenza di una buona documentazione online a proposito. Inoltre, nel pacchetto del simulatore, è presente un server, il quale ha semplificato molto la connessione al videogioco per la lettura di dati.

Assetto Corsa è stato utilizzato insieme al kit *Logitech G920 Driving Force* (volante, pedaliera e cambio) al fine di rendere più realistica l'esperienza di guida.

5.3.2 DialogFlow

DialogFlow è una delle parti più importanti del prototipo. Come affermato nel capitolo precedente permette di modellare la conversazione e gestire in maniera semplice tutte le diramazioni possibili per ogni richiesta dell'utente.

La risposta da presentare al guidatore verrà ricavata ed esposta all'utente dall'intelligenza dell'applicazione in base ai dati estratti da DialogFlow.

Come si evince dalla Figura 5.4, la console di DialogFlow consente di programmare un endpoint al quale fare richieste REST (POST in particolare) per ottenere un fulfillment, ossia una risposta da presentare a seguito dell'input ricevuto. Al fine di esporre un endpoint pubblico al quale il cloud-service potesse fare richieste HTTP è stata esposta un porta localhost tramite l'utilizzo di ngrok ².

Infine la modellazione della conversazione è stata effettuata tramite la definizione di **intenti** e **entità** per potere allenare la conversational engine nel riconoscere particolari pattern all'interno di frasi e quindi essere in grado di dedurre un contesto di conversazione e parole chiave.

²<https://ngrok.com/>. *Ultimo accesso: 06/03/2019*

⚡ Fulfillment

Webhook ENABLED

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*	<input type="text" value="https://bca23fd1.ngrok.io"/>	
BASIC AUTH	<input type="text" value="Enter username"/>	<input type="text" value="Enter password"/> ⋮
HEADERS	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	<input type="text" value="Enter key"/>	<input type="text" value="Enter value"/>
	+ Add header	
DOMAINS	<input type="checkbox"/> Enable webhook for all domains ▼	

Figura 5.4: Visione della sezione Fulfillment nella console di DialogFlow

Entità

👤 Entities CREATE ENTITY ⋮

<input type="text" value="Search entities"/> 🔍
@ azione
@ indice
@ numero
@ oggetto
@ posizione
@ presenza
@ risposta
@ spia

Figura 5.5: Visione della sezione Entities del prototipo nella console di DialogFlow

Quella delle entità è una delle due sezioni principali della console di DialogFlow. Permette la definizione dell'associazione tra parola (e sinonimi) con un valore specifico, di modo che poi l'applicazione sia in grado di capire a quale oggetto oppure a quale azione ci si stia riferendo.

Come è possibile notare dalla Figura 5.5, le entità individuabili nell'assistente sviluppato, sono state divise per categoria (azione, indice, numero, oggetto, ...), cosicché fosse più semplice per l'applicazione calcolare il corretto responso da presentare all'utente. Ogni categoria racchiude al suo interno tutte le possibili parole chiave che possono essere riconosciute, con annessi sinonimi (utili per il training della conversational engine). Un esempio è riportato nella Figura 5.6.

The screenshot shows the configuration for an entity named 'azione'. At the top right is a blue 'SAVE' button. Below it are two checkboxes: 'Define synonyms' (checked) and 'Allow automated expansion' (unchecked). A table lists the following entities and their synonyms:

accensione	accendi, accendere, accensione, accendo, accenda, accendono, accende, accendiamo, accendete, accenderlo, accenderli, accenderla, attiva, aziona
funzionamento	funziona, funzionamento, funzionano, funzionare, funzione, usare, usa, usano, usi
spegnimento	spegnimento, spegni, spegnere, spegniamo, spengono, spegnerlo, spegnerli, spegnerla, spegnerlo, spegnete, disattiva

At the bottom of the table is a link: 'Click here to edit entry'.

Figura 5.6: Visione della sezione Entities del prototipo nella console di DialogFlow

Intenti

Insieme a quella delle **entità**, quella degli **intent**i, è la sezione più importante.

Quando DialogFlow riceve una nuova frase da analizzare, compie diverse azioni

1. la confronta con tutte le frasi di fornite per ogni intento creato al suo interno al fine di trovare il miglior match;
2. una volta trovato l'intento di interesse, estrae i dati deducibili;
3. infine salva ciò che ha dedotto all'interno del **contesto**.

È proprio grazie al contesto, compilato da DialogFlow, che l'applicativo è in grado di potere capire quali siano gli oggetti e le azioni richiesti dall'utente, ma anche tenere traccia di ciò che è stato detto in precedenza durante la costruzione del responso. Questo torna utile nel caso, al fine di soddisfare correttamente la richiesta, si abbia bisogno di informazioni più specifiche e quindi si interroghi l'utente per ottenere più

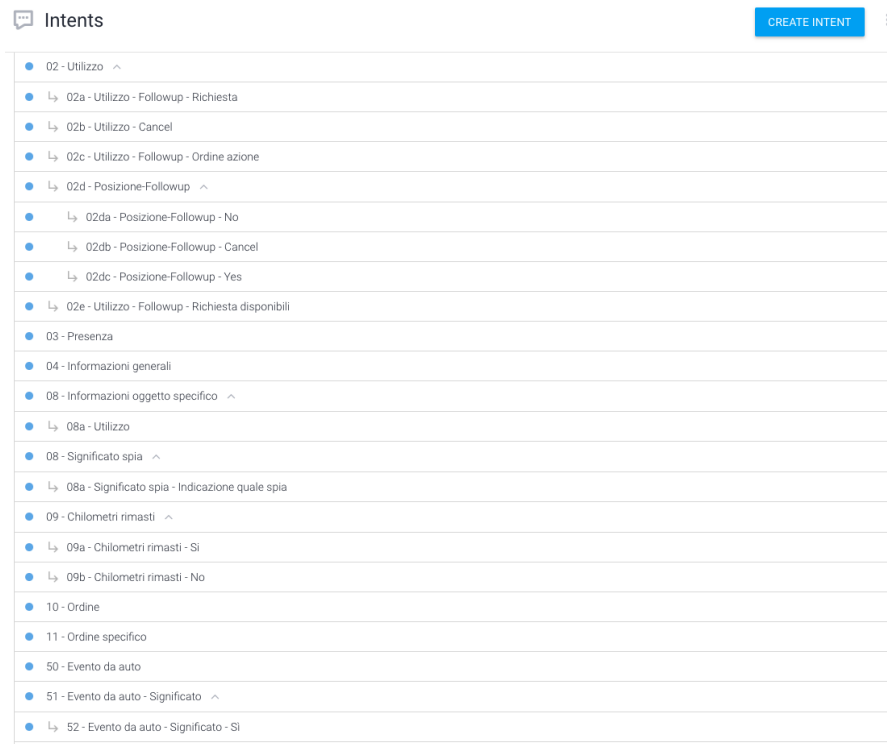


Figura 5.7: Visione della sezione Intent del prototipo nella console di DialogFlow

dettagli.

Infine all'interno è possibile definire come compilare contesto e richiesta da inviare all'applicazione, tramite due sezioni principali: training phrases, actions and parameters.

La Figura 5.8 presenta uno degli intenti scritti per l'assistente. Si può notare come siano state definite delle frasi tipo (nella sezione *Training phrases*), di modo che l'intelligenza della conversational engine sia in grado di allenarsi nel riconoscere determinati pattern. I diversi colori evidenziano particolari **entità** che si è in grado di estrarre da tali frasi. Nella sezione *Actions and parameters* si definisce l'associazione tra il valore dell'entità e la chiave associata ad esso nel payload della richiesta che verrà inviata all'applicazione. Questi dati verranno poi utilizzati per potere determinare il responso corretto da presentare all'utente.

02 - Utilizzo SAVE

☞ Come funziona il navigatore?
☞ Come funzionano i tergicristalli?
☞ Come funzionano i fendinebbia?
☞ Come funziona la radio?
☞ Come si accendono i fari fendinebbia?
☞ Come si accendono i fendinebbia?
☞ Come si accende il navigatore?

Action and parameters ^

2

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input checked="" type="checkbox"/>	azione	@azione	Sazione	<input type="checkbox"/>	Credo di non av...
<input checked="" type="checkbox"/>	oggetto	@oggetto	Soggetto	<input type="checkbox"/>	Credo di non av...
<input type="checkbox"/>	ordine	Enter entity	false	<input type="checkbox"/>	-
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	-

[+ New parameter](#)

Figura 5.8: Visione di uno specifico intento definito per il prototipo.

```

{id": "70411e30-5ed3-4f00-9164-312d2f3f568c",
"timestamp": "2019-01-24T10:07:35.787Z",
"lang": "it",
"result": {
  "source": "agent",
  "resolvedQuery": "come si accende la radio",
  "action": "2",
  "actionIncomplete": false,
  "parameters": {
    "azione": "accensione",
    "oggetto": "radio",
    "ordine": "false"
  },
  "contexts": [
    {
      "name": "utilizzo-accessorio",
      "parameters": {
        "ordine": "false",
        "oggetto": "radio",
        "azione": "accensione",
        "oggetto.original": "radio",
        "ordine.original": "",
        "azione.original": "accende"
      },
      "lifespan": 1
    }
  ]
}

```

Figura 5.9: Visione della sezione Fulfillment nella console di DialogFlow

Nella Figura 5.9 si può vedere un esempio di payload in formato JSON costruito

da DialogFlow in base a una frase ricevuta in input e inviato tramite richiesta HTTP POST all'endpoint aperto dalla nostra applicazione. In particolare i dati di maggiore interesse, al fine di elaborare una corretta risposta da presentare all'utente, risiedono:

- sotto la chiave *action*. Il numero che è possibile vedere nella Figura 5.9 permette all'applicazione di determinare in quale caso della conversazione ci si trova e quindi sfruttare i parametri ricevuti di conseguenza per costruire la risposta corretta;
- sotto la chiave *contexts* (plurale, perchè possono essere presenti contesti derivati da altri durante la conversazione, infatti il JSON inviato conterrà, in alcuni casi, un array di contesti). Ogni contesto in tale vettore, avrà assegnato un nome e una serie di parametri che corrispondono a quelli che DialogFlow è stato in grado di estrarre dalla frase ricevuto in input (nel payload sotto la chiave *resolvedQuery*);

Intenti attivati da eventi

Questa è una categoria particolare di intenti, che permettono all'assistente conversazionale di essere reattivo. Sono intenti che, anziché essere attivati da un input di un utente (una richiesta, un comando, ...), vengono attivati da un evento, lanciato, in questo caso, dall'applicazione sul veicolo. Questo permette così di raggiungere l'obiettivo desiderato di avere un'assistente non semplicemente reattivo, ma anche **proattivo**.

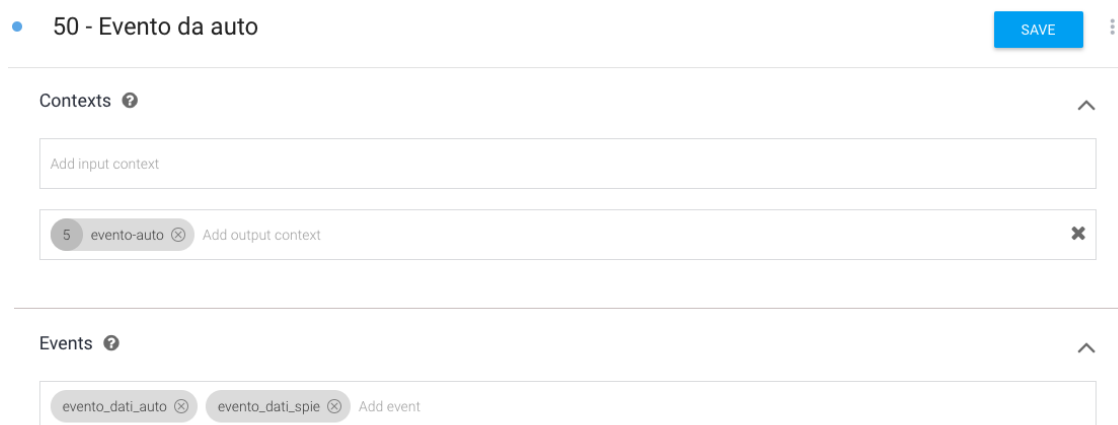


Figura 5.10: Visione di un intento attivato da un evento.

Come si può vedere dalla Figura 5.10, per creare questo tipo di intenti è stato necessario descrivere all'interno della sezione *Events* nella schermata dello specifico

intento, a quali eventi avrebbe dovuto reagire.

Una volta attivato l'evento, DialogFlow contatterà l'applicativo sul veicolo per richiedere un fulfillment (risposta da presentare all'utente). Non appena l'applicazione riceverà tale richiesta, noterà che è stata generata da un evento e quindi chiederà al guidatore le azioni da compiere. Ricevuta la risposta dall'utente, riprenderà il flusso normale di conversazione come per i classici intenti presentati al paragrafo precedente. Sono due i principali motivi per cui è stato adottato questo metodo:

1. permette di tenere la parte di interlocazione con il guidatore separata dalla logica per la determinazione di una risposta, come verrà spiegato nei paragrafi successivi;
2. consente a DialogFlow di creare già un contesto con i dati che hanno scatenato l'evento, come mostrato nella Figura 5.11.

Action and parameters ^

50

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	speed	Enter entity	#evento_dati_auto.speed	<input type="checkbox"/>
<input type="checkbox"/>	abs_enabled	Enter entity	#evento_dati_auto.abs_enabled	<input type="checkbox"/>
<input type="checkbox"/>	spia	Enter entity	#evento_dati_spie.spia	<input type="checkbox"/>
<input type="checkbox"/>	stato_spia	Enter entity	#evento_dati_spie.stato_spia	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

[+ New parameter](#)

Figura 5.11: Visione di come inserire all'interno del contesto generato da un intento i dati provenienti dall'evento.

5.4 L'applicazione sul veicolo

In questa sezione verrà esposto come è stata organizzata l'applicazione Java che è installata a bordo del veicolo. Ogni sotto-paragrafo ha lo scopo di presentare nel dettaglio il ruolo di ogni package e il funzionamento delle classi contenute in essi. Nella Figura 5.12 è possibile avere una visione di base della suddivisione dei package del progetto. Il nome dei packages rispecchia quello dei moduli rappresentati in Figura 5.1.

Prima di iniziare la discussione di ciascun modulo è necessario fare una premessa, presentando ciò accade all'avvio dell'applicativo.

L'applicazione è stata sviluppata **multithread**. Tutti i thread vengono avviati al boot dell'applicazione e ciascuno di essi avrà un compito specifico:

- serverThread: sarà il thread responsabile di gestire le richieste REST inviate da DialogFlow, quindi dovrà, in base al contesto fornire un responso da presentare all'utente. Inoltre gestirà l'aggiornamento corretto dei dati da visualizzare sulla dashboard integrativa;
- clientThread: questo thread avrà una duplice funzione ascoltare le richieste dell'utente, ma anche avvisare l'utente di eventuali situazioni critiche previa notifica da parte del *telemetryThread*;
- telemetryThread: si occuperà della lettura dei dati forniti da Assetto Corsa, inoltre chiamerà le necessario funzioni per verificare l'eventuale occorrenza di situazioni critiche.

Ogni thread al fine di svolgere le azioni desiderate utilizzerà le classi contenute in uno o più moduli.

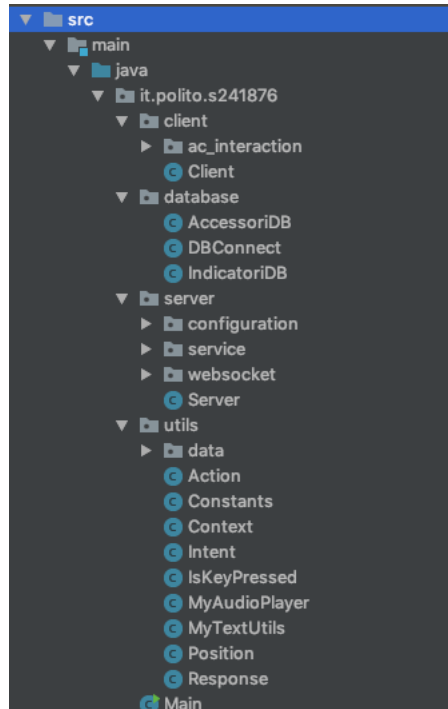


Figura 5.12: Visione dell'organizzazione dei packages all'interno dell'applicativo.

5.4.1 Modulo Server

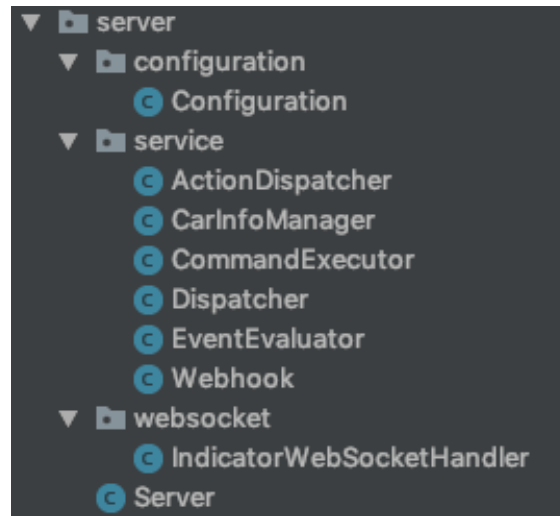


Figura 5.13: Visione dell'organizzazione interne del package server.

Come si può notare nella Figura 5.13, all'interno del package *server*, sono contenuti alcuni sottopackage tra quali sono divise le classi in base ai loro compiti:

- package configuration: qui è contenuto tutto ciò che riguarda la configurazione dell'applicazione;
- package service: qui risiede tutto ciò che riguarda la logica dell'applicazione, ossia elaborazione dei dati e intelligenza;
- package websocket: qui risiedono le classi responsabili della gestione della comunicazione con le applicazioni connesse al web socket. In particolare la dashboard integrativa.

Questo modulo è quello responsabile dell'interazione con DialogFlow. L'interazione tra i due avviene tramite richieste HTTP POST fatte all'URL esposto tramite ngrok. Il framework utilizzato per creare delle REST API è stato Spark³. Una volta ricevute queste richieste, la logica interna estrae i dati ricavati da DialogFlow e tramite una serie di switch/case annidati elabora il responso corretto da presentare all'utente.

Nel caso in cui fosse necessario, questo è il componente responsabile di richiamare le funzioni adeguate per l'aggiornamento dei dati all'interno del database o lo svolgimento di determinate azioni richieste dall'utente.

³<http://sparkjava.com/>. *Ultimo accesso: 07/03/2019*

Una particolare menzione va fatta per il package `configuration`, infatti esso offre le funzioni necessarie per autenticarsi con i servizi cloud di Google per la traduzione text-to-speech e speech-to-text.

5.4.2 Modulo per lettura della telemetria

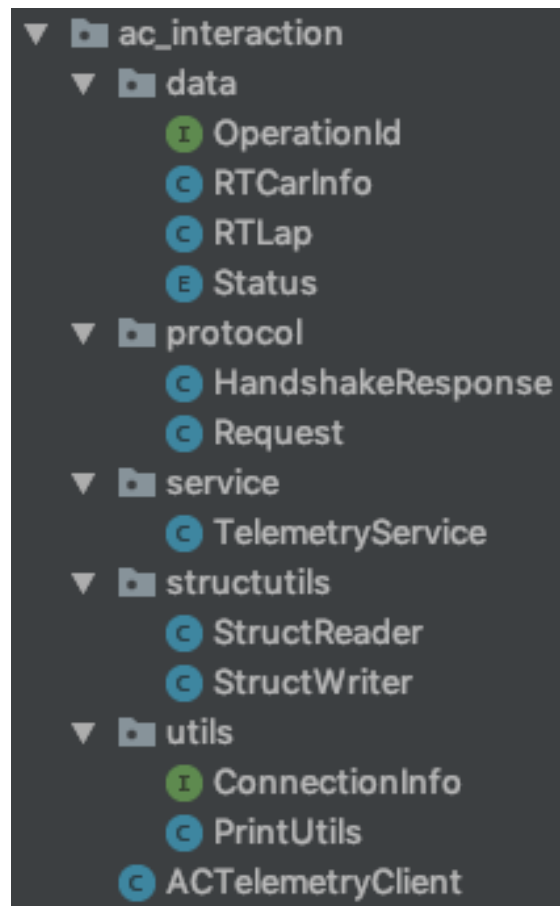


Figura 5.14: Visione dell'organizzazione del package `ac_interaction`.

Il package `ac_interaction`, come è possibile vedere dalla Figura 5.14, contiene tutte le classi necessarie per l'interazione con Assetto Corsa. È stato organizzato in maniera tale da potere essere utilizzato anche come libreria stand-alone, per leggere solamente i dati dal simulatore. Per la scrittura delle varie classi si è tradotto, esteso e corretto alcune librerie già presenti su Github, ma per altri linguaggi di programmazione ([12], [13], [14]).

La connessione al server di Assetto Corsa al fine di ottenere i dati telemetrici del veicolo richiede una sequenza precisa di azioni:

1. avviare, sulla macchina che ospita il simulatore di guida, il server responsabile della gestione della corsa che verrà monitorata per l'estrazione dei dati del veicolo;
2. avviare il videogioco e partecipare a tale corsa in LAN. A questo punto il server della gara sarà in grado di pubblicare dati riguardanti il veicolo partecipante sulla porta UDP 9996;
3. per potere notificare il server di essere interessati alla ricezione di aggiornamenti sul veicolo, l'applicazione, dovrà prima di tutto identificarsi e poi registrarsi presso di esso per uno specifico tipo di aggiornamento, come spiegato nel paragrafo *Registrazione presso Assetto Corsa* di questa sotto-sezione;
4. a questo punto finché il socket rimarrà aperto, l'applicazione resterà in ascolto dei dati, come specificato nel paragrafo *Ricezione dei dati da Assetto Corsa* di questa sotto-sezione.

Tutte le operazioni necessarie all'applicativo per essere in grado di ricevere dati dal simulatore sono contenute all'interno della classe **TelemetryService.java**, che contiene diverse funzioni per svolgere le differenti operazioni.

Nei paragrafi successivi verranno spiegate le diverse operazioni effettuate.

Registrazione presso Assetto Corsa

Dopo avere inizializzato le informazioni necessarie alla connessione (indirizzo IP del server e porta alla quale connettersi), l'applicazione dovrà registrarsi presso il server ospitante la gara per ricevere uno specifico tipo di aggiornamento.

Il server di Assetto Corsa è in grado di fornire due tipi di aggiornamento [15]:

- SUBSCRIBE_SPOT: è un update inviato solamente quando il veicolo raggiunge punti focali del percorso, ad esempio la fine di un giro di pista;
- SUBSCRIBE_UPDATE: le applicazioni registrate riceveranno aggiornamenti su tutti i veicoli in gara ogni volta che viene effettuato uno spostamento fisico della macchina.

Per potersi registrare, però, è necessario effettuare prima un'operazione di **handshake** al fine di identificarsi come applicativo. La particolare struttura dati da inviare si trova nel package *protocol* ed è la classe **Request.java**.

Dipendentemente dal tipo di sottoscrizione per la quale ci si è registrati si otterranno strutture dati diverse. In particolare:

- per SUBSCRIBE_SPOT la struttura dati ricevuta sarà del tipo **RTLap**;
- per SUBSCRIBE_UPDATE la struttura dati ricevuta sarà del tipo **RTCarrInfo**;

Tutte le classi necessarie alla lettura dei dati sono contenute nei packages *data*, *structutils* e *utils*.

Ricezione dei dati da Assetto Corsa

Nonostante ci sia la possibilità di registrarsi per due diversi tipi di aggiornamenti, il prototipo è interessato solamente nel ricevere dati per una `SUBSCRIBE_UPDATE`, ossia dati continui sullo stato del veicolo. L'applicazione, in questo caso, continuerà a leggere dati in arrivo sul socket connesso al sever di Assetto Corsa fino alla sua chiusura.

La sequenza di operazioni effettuate al fine di ricevere i dati telemetrici e passarli all'applicazione principale è la seguente:

1. viene allocato un buffer per la ricezione dei dati esattamente della dimensione delle informazioni che si vogliono ricevere;
2. si riceve un pacchetto UDP dal socket connesso al server di Assetto Corsa;
3. viene creato un oggetto di tipo `RTCarInfo` che racchiude tutte le informazioni sul veicolo, partendo dai byte ricevuti sul socket (per la lettura e traduzione dei byte in strutture dati Java vengono utilizzate le classi che risiedono nel package *structutils*);
4. infine viene invocato il metodo della classe `CarInfoManager.java` per controllare le informazioni ricevute e aggiornare i dati sul veicolo salvati all'interno dell'applicazione.

5.4.3 Modulo Client

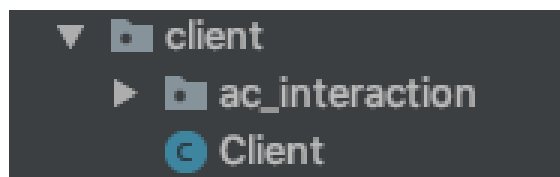


Figura 5.15: Visione dell'organizzazione del package *client*.

Il package *client* può essere considerato come un wrapper del package *ac_client* con l'aggiunta della classe `Client.java`, responsabile della maggior parte dei compiti inerenti l'interazione con il guidatore. Tali compiti consistono in:

- ascolto delle richieste dell'utente;
- trasformazione della richiesta da formato audio a formato testuale e invio della stessa a `DialogFlow`;

- traduzione della risposta da formato testuale a formato audio;
- riproduzione dell'audio corrispondente alla risposta.

Ricezione richieste guidatore

Per ricevere la richiesta dell'utente, il thread al cui interno viene lanciato il processo client, resterà in ascolto sulla linea audio del dispositivo sul quale è installato il microfono ricevente, per dei dati. Onde evitare di ricevere dati indesiderati, l'inizio dell'ascolto vero e proprio viene regolato dalla classe **IsKeyPressed.java**.

Una volta ricevuto un input dall'utente, il client invia la richiesta per la traduzione da audio a testo solamente quando risulta che il tasto per l'avvio dell'ascolto è stato premuto. Nel caso specifico del prototipo si considera come tasto per avviare l'ascolto la barra spaziatrice dell'elaboratore sul quale viene fatta girare l'applicazione.

Trasformazione delle richieste da audio a testo

Per la trasformazione da audio a testo, vengono utilizzati dei cloud-services di Google. Per questo motivo all'avvio del thread client per l'ascolto delle richieste del guidatore verrà inizializzato un oggetto **SpeechClient** per l'interazione con tali servizi. In particolare tale tipo di client è asincrono, per cui funzionante tramite callbacks e **Observers**. Per questo motivo una volta creato e inizializzate tutte le credenziali/variabili (sample rate, lingua, ...), viene assegnato ad esso un oggetto di tipo **ResponseObserver<StreamingRecognizeResponse>**, il quale ha il compito di definire il comportamento da adottare nel caso di un response da parte del cloud-service. La libreria per l'interazione con i cloud-services fornisce 4 tipi di callbacks, ma per il prototipo è stato sufficiente l'utilizzo di tre di esse:

- *onStart*: è possibile utilizzare questa funzione per definire operazioni di setup per il controllo del flusso di dati (azione che per ciò che è stato sviluppato, non era necessaria);
- *onResponse*: questa callback viene invocata quando il cloud-service produce un responso (traduzione da audio a testo). Una volta estratto il risultato, viene resettato il bottone di ascolto, per indicare che avverrà la riproduzione di un audio e quindi l'assistente non dovrà ascoltare. In seguito viene inviata a DialogFlow una richiesta contenente la trascrizione, di modo da ottenere la risposta da presentare all'utente;
- *onComplete*: è la callback chiamata alla chiusura del flusso di dati con il webservice, in teoria non sarebbe necessario, ma in questo caso salviamo lo storico di tutte le trascrizioni ottenute in precedenza;

- *onError*: come si può capire dal nome, questa è la callback invocata nel caso di errore.

Il compito, forse, più importante del quale si deve prendere carico il modulo client è quello di mantenere sincronizzati i contesti della conversazione estratti da DialogFlow (e salvati dal serverThread nell'applicazione) e allegarli ad ogni nuova richiesta dell'utente, cosicché, in caso la conversazione sia già stata avviata, DialogFlow sia in grado di recuperare da dove si era lasciato. Questo è necessario perché il thread contenente il modulo server e quello contenente il modulo client lavorano separatamente.

Dopo avere ricevuto la risposta da presentare all'utente, è necessario trasformarla in audio e avviarne la riproduzione, tutto ciò è effettuato dalla classe **MyAudioPlayer.java**.

Trasformazione delle richieste da testo ad audio

Al fine di eseguire questa operazione è necessaria l'interazione con l'altro cloud-service di Google anticipato nella sezione 4.1.1, quello per la traduzione da testo a audio.

Il procedimento è molto simile a quello presentato in precedenza per la situazione inversa, solamente che in questo caso non vi è bisogno di specificare callbacks particolari. Dopo avere configurato con i parametri corretti l'oggetto **TextToSpeechClient**, che servirà per interagire con il cloud-service, sul socket aperto per la connessione verrà ricevuto uno stream di byte corrispondente al contenuto testo inviato per la traduzione, trasformato in audio. Una volta ricevuti tutti i byte, l'applicazione scriverà in un file audio il risultato e ne avvierà la riproduzione. Infine, per evitare di riempire la memoria a disposizione, il file audio verrà eliminato dopo la sua riproduzione.

5.4.4 Modulo per interazione con cruscotto integrativo

Questo paragrafo ha lo scopo di spiegare la gestione del flusso di dati passato alla dashboard integrativa lato server, successivamente verrà discusso come avviene la visualizzazione dei dati all'interno dell'applicazione Angular (lato client).

Come anticipato nel paragrafo 5.4.1, all'avvio dell'applicativo viene invocato un metodo per la configurazione di un endpoint sul quale verrà aperto un web socket. Tutto questo è effettuato tramite il metodo statico *webSocket* della libreria di Spark che comunica al framework che un determinato path (nel caso del prototipo **/socket/indicators**) deve essere utilizzato per connessioni attraverso web socket. In questa maniera si può garantire che tra l'applicazione registratasi presso quell'endpoint e l'applicazione server vi sia un flusso continuo di dati, cosa necessaria per potere ottenere il display dei dati ricevuti da Assetto Corsa in tempo reale sulla dashboard integrativa.

Una particolare classe sarà designata alla modellazione dei dati che sono in grado di viaggiare su questo canale di comunicazione. Tale classe dovrà essere annotata come `@WebSocket`, cosicché il framework sia in grado di riconoscerla e dovrà avere metodi annotati a seconda del momento, nel ciclo di vita del `WebSocket`, nel quale dovranno essere invocati.

I metodi forniti dal framework per la gestione della comunicazione e il ciclo di vita del web socket sono tre:

1. *onConnect*: questa funzione viene invocata non appena un'applicazione client si connette al web socket, di conseguenza, non avendo nessun dato ancora di cui fare il display, le verranno inviate tutte le informazioni sugli indicatori salvati all'interno del database lato server insieme alle informazioni del veicolo ottenute da Assetto Corsa (in questo caso la velocità attuale, la marcia inserita e il numero di giri al minuto);
2. *onClose*: questa funzione viene chiamata quando un determinato listener (applicazine client connessa) si disconnette dal web socket, per cui lato server lo si dovrà semplicemente eliminare dalla lista delle applicazioni connesse;
3. *onUpdate*: questa funzione è invocata quando viene ricevuto, da un determinato listener, un update sullo stato dei dati da esso contenuti. Nel particolare caso del nostro prototipo, un update sullo state delle spie.

5.4.5 Modulo per interazione con la base dati

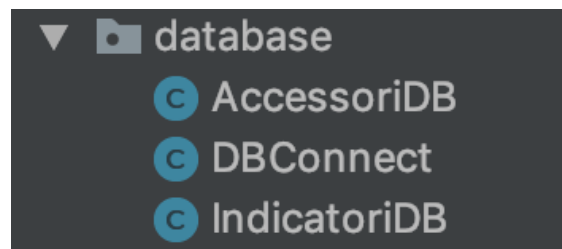


Figura 5.16: Visione dell'organizzazione del package database.

Ultimo, ma non per importanza è il package responsabile della gestione della base dati. Come database è stato selezionato un database **SQLite**. Il motivo di questa scelta è quello che questo tipo di database è facilmente incorporabile all'interno di applicazioni, soprattutto se esse devono girare su centraline con poca potenza computazionale. Le classi all'interno di questo package hanno due principali compiti:

1. ottenere informazioni a proposito dei vari accessori disponibili sul veicolo;

2. tenere aggiornate tali informazioni coerentemente con lo stato del veicolo e i comandi dell'utente.

Per connettersi alla base dati viene utilizzato un oggetto di tipo **DBConnect**, il quale ha il semplice scopo di fornire una connessione con la base dati. Per tale classe è stato utilizzato un pattern *singleton* (che consiste nell'avere una istanza statica dell'oggetto all'interno della classe stessa, definire il costruttore privato e permettere l'accesso a tale istanza solo tramite metodi statici), di modo da avere un unico oggetto per tutta l'applicazione tramite il quale accedere alle risorse salvate nel database.

La base dati è stata organizzata con due tabelle: una tabella per le informazioni degli accessori e un'altra per tenere traccia delle spie associate a tali accessori con relativo stato (accesa/spenta). In particolare, per potere dedurre a quale accessorio sia associata una particolare spia è utilizzato, come *foreign key*, l'id (attribuito assegnato dal database) dell'accessorio, nel record della relativa spia.

Per l'accesso ai dati sono utilizzate le classi **AccessoriDB.java** (per eseguire queries alla tabella degli accessori) e **IndicatoriDB.java** (per ottenere informazioni sulle spie). Ciascuna di queste classi contiene al suo interno metodi per eseguire queries al fine di ottenere informazioni particolari dalla relativa tabella.

Le azioni compiute dai metodi per eseguire una query sono le seguenti:

1. definizione della query;
2. connessione con il database (tramite un oggetto di tipo **DBConnect**);
3. ottenimento del risultato;
4. estrazione dei dati dal risultato;
5. restituzione del risultato della struttura desiderata.

Come riferito nel Capitolo successivo 7, l'assistente non presenta molta varietà nella struttura delle frasi delle risposte presentate al guidatore. Questo è anche dovuto al fatto che nel database sono salvati anche alcuni response a domande dell'utente, che risultano di conseguenza *hard-coded*. Ad esempio, se il guidatore chiede all'assistente il funzionamento di un determinato accessorio, quest'ultimo risponderà sempre allo stesso modo esponendo come frase quella che è in grado di reperire dalla base dati.

5.5 Dashboard

In quest'ultima sezione verranno spiegati gli aspetti principali dell'applicazione Angular ⁴ sviluppata per visualizzare le informazioni estratte da Assetto Corsa e

⁴<https://angular.io/>. *Ultimo accesso*: 08/03/2019

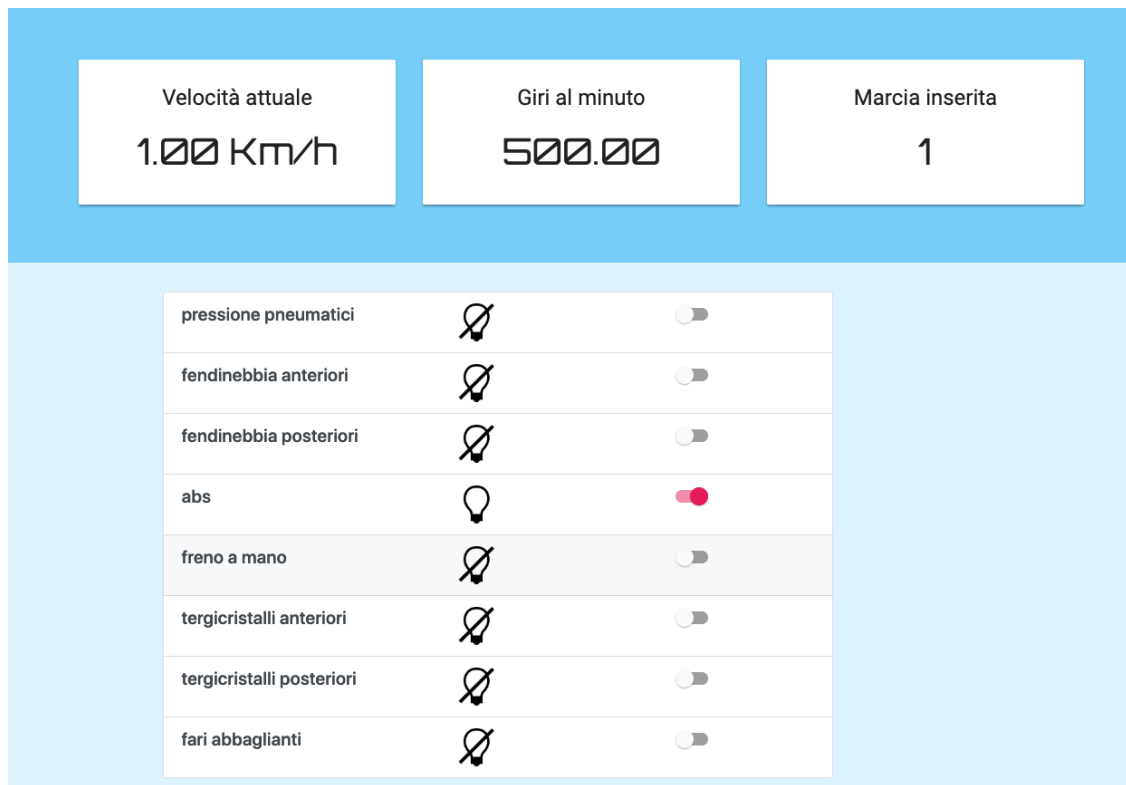


Figura 5.17: Visione della dashboard come appare all'utente.

le spie del veicolo sulle quali è possibile agire modificandone lo stato, essendo la dashboard di Assetto Corsa mancante di tali indicatori.

Come si può notare nella Figura 5.17, il cruscotto integrativo, è suddiviso in due parti:

1. nella parte alta sono visualizzati i dati ricavati da Assetto Corsa, in particolare velocità corrente del veicolo, numero di giri al minuto del motore e marcia inserita;
2. nella parte bassa vi è una serie di indicatori presenti nel database dell'applicazione, a ciascuno di essi è stato associato un bottone in modo da poterne cambiare lo stato e testare particolari situazioni di interesse come ad esempio quella presentata nei casi d'uso al Capitolo 3.

Essendo un'applicazione Angular, si è ragionato in termini di componenti e servizi:

- i componenti sono responsabili della corretta visualizzazione dei dati ottenuti dal server (in questo caso, gli indicatori e i dati di Assetto Corsa);

- i servizi sono responsabili di contattare il server per la ricezione dei dati, inviare un aggiornamento e fornire le informazioni necessarie ai componenti per la visualizzazione.

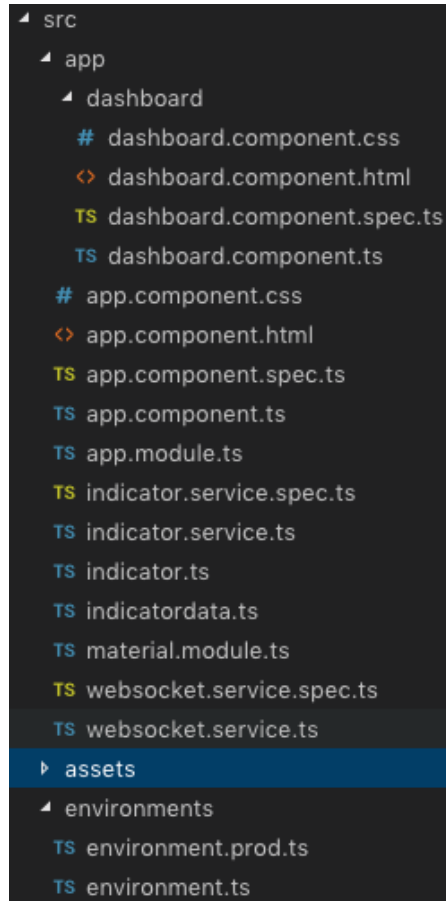


Figura 5.18: Visione dell'organizzazione dell'applicazione per la dashboard integrativa.

Come si evince dalla figura 5.18, l'applicazione è composta da un singolo componente responsabile della visualizzazione dei dati ricevuti dai servizi. In particolare sono stati definiti due tipi di servizi per questa dashboard:

1. un servizio responsabile della connessione con il web socket aperto dall'applicazione centrale in modo da ottenere tutti i dati riguardanti gli indicatori e le informazioni ottenute da Assetto Corsa;
2. un servizio responsabile del parsing dei dati ricevuti sul canale di comunicazione in formato JSON.

Al fine di ottenere una comunicazione bidirezionale è stato impiegato l'uso dei **Subject**<T> che sono particolari strutture dati che consentono di specificare le

azioni da compiere sia nel caso vi sia una modifica locale e quindi il bisogno di notificare il server, che viceversa.

5.6 Casi d’uso e interazione dei componenti

In questa sezione verrà presentata l’interazione dei vari componenti in alcuni esempi di casi d’uso presi dal capitolo 3. Verrà descritto un esempio dove l’interazione è iniziata dall’utente richiedendo l’esecuzione di un comando specifico, un esempio dove l’interazione è iniziata dall’utente per la richiesta di informazioni sul veicolo e un ultimo esempio dove l’interazione invece viene iniziata dall’assistente. Le azioni compiute dall’assistente verranno suddivise in fasi e per ciascuna di esse verranno descritti i componenti che entrano in gioco e i loro compiti.

5.6.1 Da guidatore: Comando

Descrizione del caso d’uso

Contesto: Si sta guidando in condizioni di poca visibilità (nebbia).

Guidatore: “Accendi i fendinebbia”

Veicolo: “Quali?”

Guidatore: “Anteriori”

Veicolo: “Azione effettuata”

Moduli coinvolti

Come mostrato in Figura 5.19 il primo componente ad entrare in azione è il modulo *client* responsabile di rimanere in ascolto di eventuali input del guidatore. Tale componente, all’avvio dell’applicativo, compie due azioni principali:

1. inizializza un client (**SpeechClient**) per interagire con il cloud service responsabile della traduzione da input vocale a input testuale. In questo passaggio vengono specificate delle callbacks per specifici eventi che possono accadere nel processamento della richiesta inviata descritte in precedenza nella Sezione 5.4.3;
2. inizializza un oggetto **DataLine** per specificare tutte le informazioni audio necessarie per estrarre l’input ricevuto dal microfono.

Una volta che sulla linea di ascolto è ricevuto un input, ad esempio “Accendi i fendinebbia”, il modulo innanzitutto controlla che non vi sia la riproduzione di

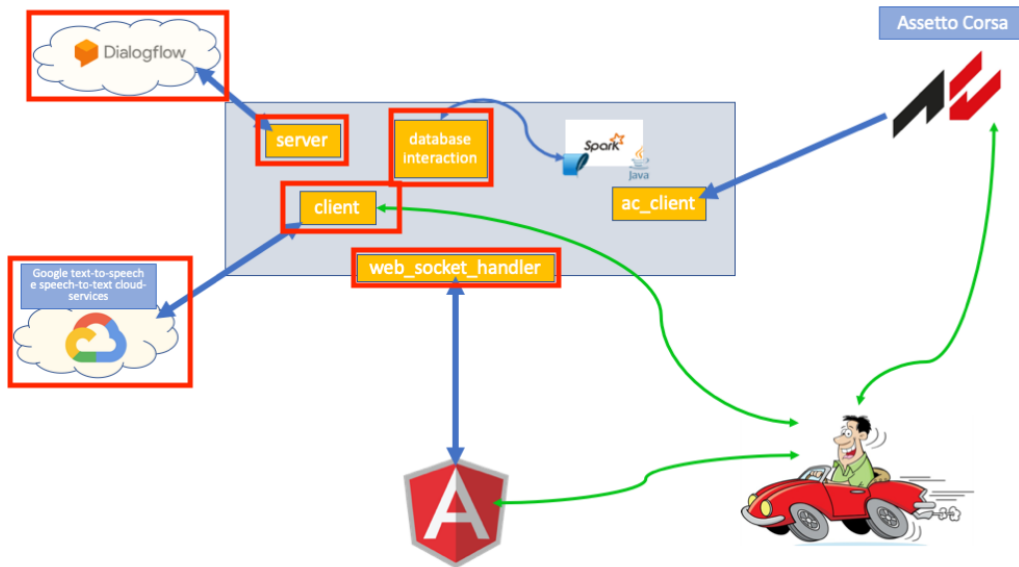


Figura 5.19: Architettura del sistema con evidenziati i componenti utilizzati per questo caso d'uso.

qualcos'altro in corso. In caso l'assistente stesse riproducendo qualcosa all'utente, allora l'input appena ricevuto viene scartato (per evitare effetti di tipo eco). Nel caso contrario invece si procede con il passo successivo e si riceve l'input dal buffer del microfono. Prima di inviare la richiesta per la traduzione da vocale a testuale, al fine di capire se i dati appena ricevuti siano validi, viene controllato che il bottone per avviare l'ascolto sia stato effettivamente premuto. Se così non fosse, i dati vengono scartati, perchè potrebbero essere semplicemente rumore di fondo oppure ciò che il microfono è riuscito a ricevere di una conversazione avvenuta all'interno dell'abitacolo, ma che non coinvolge l'assistente. Nel caso in cui invece l'ascolto sia stato comandato dall'utente, allora il client invia i byte letti dalla linea di input al cloud service per la traduzione (internamente lo **SpeechClient** utilizza un meccanismo di richieste HTTP per interagire con il servizio).

Una volta ricevuta la traduzione testuale, il client ha tutti i dati necessari per elaborare un responso da presentare al guidatore. Si apre quindi una nuova fase che coinvolge il modulo *client* e *DialogFlow*, come evidenziato nella Figura 5.19, ed ha lo scopo di ricavare il contesto della conversazione, intenti, oggetti di interesse e azioni da compiere.

Dopo avere inviato la richiesta, il client si mette in attesa di un responso da parte di *DialogFlow* contenente al suo interno la risposta da presentare all'utente. Anche in questo caso l'interazione tra i due moduli avviene tramite richieste HTTP. *DialogFlow* estrae tutte le informazioni dall'input ricevuto. Tali dati sono utilizzati

dalla logica interna dell'applicativo (contenuta all'interno del modulo *server*) al fine di costruire un responso da presentare al guidatore.

Una volta ricevuta la richiesta, da parte di DialogFlow come specificato nella Sezione 5.4.1, il componente *server* applica i relativi controlli per creare un **Fulfillment** da ritornare come risposta alla richiesta dell'utente. Per fare ciò il modulo ha bisogno di interagire con altri componenti dell'applicazione per capire se la risposta può essere immediatamente fornita oppure vi è bisogno di ulteriori informazioni da richiedere al guidatore. I moduli con i quali il componente *server* va a collaborare sono quelli che in grado di accedere a informazioni sullo stato del veicolo o sugli accessori presenti all'interno dell'auto. Nel caso particolare di questo esempio, essendo che le informazioni sull'accessorio fendinebbia non sono pervenute da Assetto Corsa direttamente, ma solo presenti nel database, il modulo con il quale si andrà a interagire è il componente *database_interaction*, come mostrato in Figura 5.19.

In questo caso specifico l'accessorio che l'utente intende attivare è presente in due posizioni diverse nel database: anteriore e posteriore. Quindi per potere svolgere l'azione di accensione correttamente, il modulo *server* ha bisogno di informazioni aggiuntive a proposito della posizione dell'accessorio a cui l'utente si sta riferendo. Di conseguenza, come **Fulfillment**, il componente ritornerà a DialogFlow la frase "Quali?" per chiedere al guidatore a quali fendinebbia si stesse riferendo.

Ricevuto il **Fulfillment**, DialogFlow, produrrà un responso per il modulo *client* contenente la risposta da presentare all'utente.

A questo punto il modulo *client* utilizzerà il cloud service per la traduzione da formato testuale a vocale e riprodurrà il risultato all'utente. In quest'ultima fase i moduli che entreranno in azione sono gli stessi della fase di ricezione dell'input, come evidenziato in Figura 5.19. La riproduzione della risposta all'utente seguirà alcuni passaggi specificati dalla classe **MyAudioPlayer**:

1. ricevuta la risposta dal DialogFlow, verrà tradotta in byte stream grazie all'utilizzo del cloud service per la traduzione speech-to-text;
2. ottenuta la traduzione verrà scritto un file audio contenente tale stream;
3. verrà riprodotto il file audio appena scritto. In questa fase verranno settati gli appositi flag per evitare che il modulo *client* consideri come input valido ciò che riceve dall'ascolto sulla linea audio del microfono. Come detto in precedenza, questo è stato una misura presa per evitare effetti di tipo eco;
4. una volta riprodotto il file audio, esso verrà cancellato per liberare memoria.

Lo stesso procedimento viene applicato per la risposta dell'utente "Anteriori" alla richiesta dell'assistente "Quali?". L'unica differenza con ciò che è stato effettuato

in precedenza è che in questo caso il modulo *server* ha a disposizione tutte le informazioni necessarie per eseguire l'azione richiesta dall'utente, infatti:

- **azione**, **intenti** e **oggetti** di interesse sono ottenuti tramite il contesto compilato da DialogFlow e mantenuto per tutta la conversazione;
- la **posizione** dell'oggetto che mancava al passaggio precedente viene inserita all'interno del contesto da DialogFlow alla seconda risposta dell'utente.

Giunti a questo punto, il modulo *server* attiva l'accessorio *fendinebbia* come comandato dal guidatore. Per fare ciò e allineare i dati contenuti in tutti i componenti dell'applicazione è necessario interagire con due moduli differenti come mostrato in Figura 5.19:

1. il componente *database_interaction* per aggiornare lo stato dell'accessorio e del suo relativo indicatore all'interno della base dati interna dell'assistente;
2. il componente *web_socket_handler* per notificare la dashboard integrativa per la visualizzazione degli accessori che l'oggetto *fendinebbia* ha cambiato il suo stato e quindi aggiornare di conseguenza l'interfaccia grafica.

Una volta effettuata l'azione richiesta l'assistente dovrà inviare un messaggio di conferma all'utente. Nel prototipo è stato definito un response di default per quando viene completata una task comandata dal guidatore ed è "Azione effettuata". Tale frase verrà definita come **Fulfillment** nel response ritornato a DialogFlow e verrà riprodotta all'utente come specificato in precedenza.

5.6.2 Da guidatore: Richiesta di informazioni (derivate)

Descrizione del caso d'uso

Contesto: Situazione di guida dove si è quasi in riserva.

Guidatore: "Ho notato che sono quasi in riserva. Quanti chilometri di autonomia mi rimangono?"

Veicolo: "Vuoi che calcoli considerando la velocità attuale?"

Guidatore: "Sì"

Veicolo: "Ti rimangono 15 chilometri di autonomia"

Moduli coinvolti

Anche in questo esempio, come quello presentato nella Sezione 5.6.1, la prima parte di interazione con l'utente coinvolge solamente il modulo *client*, i cloud-service

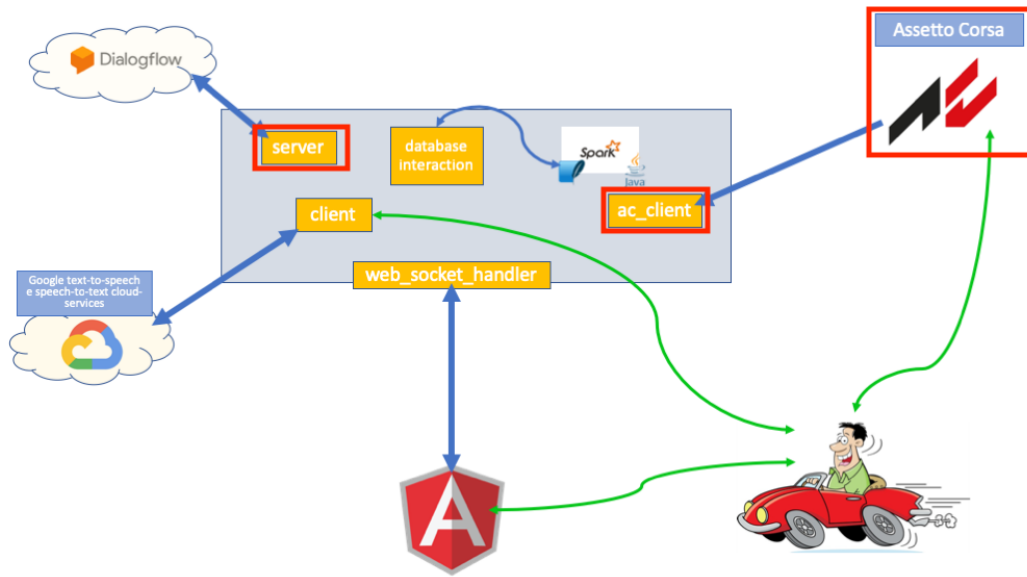


Figura 5.20: Architettura del sistema con evidenziati i componenti che agiscono nella fase di ricezione e aggiornamento dei dati sul veicolo.

per la traduzione speech-to-text e DialogFlow.

In questo caso però, la logica interna contenuta nel modulo *server*, non andrà ad interagire con i moduli responsabili della gestione delle informazioni riguardanti gli accessori, bensì interagirà con il modulo responsabile della lettura dei dati del veicolo *ac_client* come evidenziato nella Figura 5.20. In particolare quest'ultimo modulo è responsabile di ricevere costantemente informazioni da parte del simulatore sottoforma di pacchetti UDP ricevuti sul socket connesso al server che hosta la gara alla quale partecipa il veicolo da monitorare. Ad ogni nuovo dato ricevuto vengono sovrascritte delle variabili in memoria centrale che servono a tenere traccia dei dati d'interesse, nel caso specifico del prototipo: velocità corrente del veicolo, giri al minuto del motore e marcia inserita. Tali dati vengono ogni volta aggiornati sulla dashboard integrativa inviando informazioni sul web socket connesso ad essa.

Nel caso d'uso specifico considerato il modulo *server* utilizzerà i dati pushati in memoria dal modulo *ac_interaction* per calcolare il numero di chilometri di autonomia rimasti per il veicolo e produrrà una risposta con tale informazione. Anche in questo caso il response è hard-coded, per cui l'unica cosa che può variare se chiesto lo stesso tipo di informazione più volte è il numero dei chilometri rimasti, non la struttura della frase.

La presentazione della risposta al guidatore avverrà nello stesso modo presentato per l'esempio precedente, nella Sezione 5.6.1.

di una spia particolare. La gestione delle informazioni riguardanti spie e accessori è a carico dei moduli *database_interaction* e *web_secoket_handler* (quest'ultimo per la visualizzazione dei dati a video) come mostrato in figura 5.21. Quando questi due moduli notano un cambiamento nello stato di una spia che potrebbe segnalare qualche problema con il veicolo, comandano al modulo server di notificare il guidatore tramite avviso vocale. Essendo che per sostenere una conversazione vi è bisogno di stabilire un contesto che DialogFlow, in seguito, andrà a popolare di tutte le informazioni ottenibili da parte dell'utente, una volta che i due moduli si sono accorti del cambiamento di stato lanciano un evento in DialogFlow al quale segue una chiamata POST all'endpoint esposto per il **FulFillment** come negli altri casi. A questo punto il contesto della conversazione contiene solamente le informazioni riguardanti l'indicatore che ha cambiato stato. Il modulo *server*, incaricato di determinare il responso ad ogni richiesta ricevuta da DialogFlow, può eseguire due azioni:

1. nel caso in cui la spia non indica nessun problema con il veicolo in sè non esegue nulla;
2. nel caso contrario in cui la spia, invece, può indicare un problema all'interno dell'auto riproduce un avviso al guidatore.

Nella situazione specifica dell'esempio, la spia degli pneumatici, può indicare una pressione troppo bassa delle gomme e questo è un motivo sufficiente per avvisare il guidatore. Di conseguenza, come nel caso del modulo *client* viene utilizzata la class **MyAudioPlayer** per riprodurre il messaggio “La luce pressione degli pneumatici è accesa”. Le operazioni compiute da tale classe sono identiche a quelle degli altri due esempi.

Una volta che il messaggio è stato ricevuto dall'utente, quest'ultimo può non rispondere concludendo quindi la conversazione a questo punto, altrimenti in caso di richieste da parte del guidatore (come ad esempio “Che cosa significa?”), la conversazione si svilupperà nella stessa maniera presentata per i casi precedenti. L'unica differenza tra questo esempio e quelli presentati nella Sezione 5.6.1 e nella Sezione 5.6.2 è il preambolo dove i moduli *database_interaction* e *web_secoket_handler* scatenano un evento per attivare DialogFlow e notificare che vi è stato un cambio di stato all'interno del veicolo.

Capitolo 6

Risultati

In questo capitolo verranno esposti i test effettuati sull'assistente sviluppato per questa tesi. Prima di tutto verrà fatta una panoramica delle specifiche degli elaboratori sui quali sono state dislocate le varie parti dell'applicazione, dopodichè verrà presentato l'effettivo test funzionale al quale è stato sottoposto il prototipo evidenziando il suo comportamento in situazioni particolari. Infine saranno descritte le latenze di risposta dell'assistente e come sono state calcolate.

6.1 Specifiche degli elaboratori

6.1.1 Elaboratore simulatore

Le specifiche dell'elaboratore sul quale è stato installato e utilizzato Assetto Corsa sono disponibili in Figura 6.1. Tale computer presenta risorse limitate, ma sufficienti ad ottenere un'esperienza di guida fluida. Si volesse migliorare l'esperienza di guida bisognerebbe fare un upgrade di tale elaboratore. L'elaboratore possiede un processore Intel Core i7 (2.70 GHz), 8.00 GB di RAM e scheda video integrata. Quello che influisce di più sull'esperienza di guida è il fatto che il computer non possieda scheda video dedicata; attivare quindi server per lettura della telemetria e videogiochi risulta pesante per tale elaboratore.

6.1.2 Elaboratore assistente

L'elaboratore sul quale è stato installato l'assistente presenta le specifiche hardware visualizzate nella Figura 6.2.

Come è possibile notare dalla Figura 6.2 l'elaboratore presenta sufficienti risorse per il deploy senza problemi dell'applicazione dell'assistente (16 GB di RAM, processore Intel Core i7 4 Core (2.80 GHz)). Ad ogni modo, al di là dell'uso di Java, sono state utilizzate tecnologie non troppo pesanti per l'elaboratore in grado di favorire un trasporto futuro su centraline con risorse minori (Spark, SQLite).

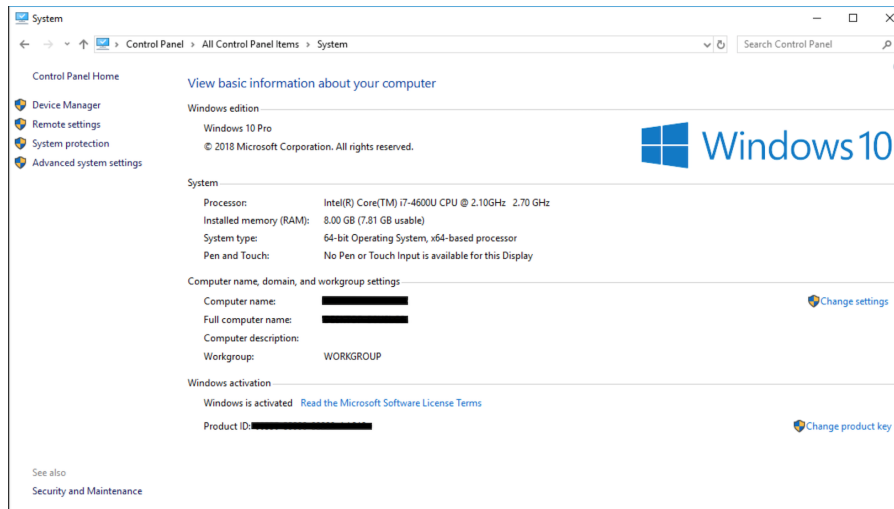


Figura 6.1: Specifiche dell'elaboratore utilizzato per il deploy dell'assistente.



Figura 6.2: Specifiche dell'elaboratore utilizzato per il deploy dell'assistente.

Per la comunicazione con i web services è stata utilizzata una connessione Wifi. La scheda di rete di tale elaboratore presenta invece le specifiche visualizzate nella Figura 6.3.

La velocità del collegamento, come si evince dalla Figura 6.3 è di 144 Mbit/s e tale bandwidth è più che sufficiente per la comunicazione con i cloud services essendo che i dati passanti sul canale non sono di grande dimensione (JSON, byte stream).

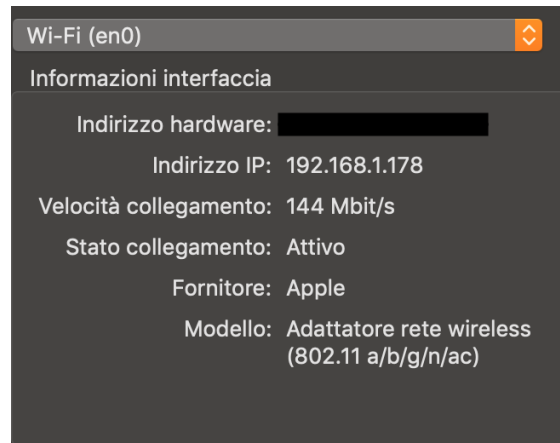


Figura 6.3: Specifiche della velocità di trasmissione della scheda di rete dell'elaboratore.

6.2 Test funzionale

Il corretto funzionamento dell'assistente è stato testato in base ai casi d'uso definiti nel Capitolo 3.

Un ricercatore ha verificato in una prima fase la correttezza delle risposte dell'assistente in base alle richieste dell'utente, seguendo rigorosamente la conversazione così come è stata impostata nel caso d'uso.

In una seconda fase invece sono state apportate delle modifiche lessicali alle richieste presentate dall'utente ed è stato verificato il corretto comportamento dell'assistente in base a tali variazioni.

In un'ultima fase è stata testata la capacità dell'assistente di cambiare contesto di conversazione nel mezzo della conversazione stessa, ad esempio: si è iniziata la conversazione così come definita per un caso d'uso e nel mezzo della stessa, prima di arrivare alla conclusione, si è cambiato ad un altro caso d'uso verificando che l'assistente si adattasse di conseguenza.

6.2.1 Risultati test funzionale

Prima fase: no variazioni

Durante la prima fase un ricercatore ha testato il comportamento dell'assistente utilizzando i casi d'uso così come definiti nel Capitolo 3, senza apportare alcuna variazione alla conversazione.

In questa fase, è stato definito come successo il fatto che l'assistente riproducesse la risposta attesa. Il prototipo si è rivelato correttamente funzionante per il 100% dei casi d'uso descritti nel Capitolo 3 quando alla conversazione non viene applicata alcuna variazione.

Seconda fase: variazioni lessicali

Nella fase successiva, un ricercatore, ha testato il corretto comportamento dell'assistente quando alle richieste proposte dall'utente sono applicate variazioni lessicali, la struttura della conversazione ha comunque seguito quella dei casi d'uso descritti nel Capitolo 3.

In questa seconda fase, è stato definito come successo il fatto che l'assistente riuscisse a portare a termine la conversazione nella stessa maniera della fase precedente (condizione ideale).

Le variazioni sono possibili su due livelli: *vocaboli* e *struttura della frase*. Per quanto riguarda le variazioni a livello di vocaboli, sono stati testati cambiamenti alle richieste dell'utente come quelli visibili nella Tabella 6.1. In tale tabella sono contenuti tutti i vocaboli corrispondenti agli accessori presenti nel database, per i quali l'utente può chiedere l'accensione o lo spegnimento. Invece per quando concerne le variazioni lessicali a livello di struttura della frase sono stati testati dei cambiamenti che modificassero la frase detta in input, mantenendone però il concetto generale e variazioni che fornissero abbastanza informazioni all'assistente per potere eseguire direttamente il comando. Nella Tabella 6.2 sono presenti esempi di frasi utilizzate per il test.

Al fine di verificare il corretto comportamento dell'assistente sono state apportate

Vocabolo	Alternativa
Fendinebbia	Fari fendinebbia
Abbaglianti	Fari abbaglianti
Tergicristalli	-
Radio	Musica, Lettore CD

Tabella 6.1: Tabella contenente vocaboli testati e le loro alternative

modifiche ai casi d'uso presentati nel Capitolo 3, sostituendo gli accessori con i vocaboli alternativi presenti nella Tabella 6.1. Il numero di test effettuati è pari al numero di alternative per l'accessorio considerato.

Frase di riferimento	Variazione	Effetto
Come si accendono i fendinebbia?	Come si accendono i fendinebbia anteriori/posteriori?	L'assistente esporrà direttamente le istruzioni per l'accensione dei fendinebbia anteriori/posteriori
Accendi i fendinebbia	Accendi i fendinebbia anteriori/posteriori	L'assistente eseguirà direttamente il comando di accensione per i fendinebbia anteriori/posteriori
Accendi i tergicristalli	Accendi i tergicristalli anteriori/posteriori	L'assistente eseguirà direttamente il comando di accensione per i tergicristalli anteriori/posteriori
Ho notato una spia accesa sul cruscotto e non so che cosa significa	Cosa significa quella spia accesa sul cruscotto	L'assistente esporrà una lista di descrizioni di spie accese attualmente sul cruscotto e l'utente dovrà poi indicare a quale si riferisce
Che cosa significa?	Cosa vuol dire? - Qual è il significato?	L'assistente esporrà il significato della spia alla quale ci si sta riferendo al momento (utilizzato per avvisi e alert critici soprattutto)
Quanta autonomia mi rimane?	Calcola autonomia residua - Sono quasi in riserva, quanti chilometri di autonomia mi rimangono?	L'assistente calcolerà l'autonomia residua in base ai dati posseduti e la esporrà all'utente
Quali accessori sono presenti in questa macchina?	Che optional sono presenti in questa macchina? - Ci sono accessori?	L'assistente restituirà una lista di accessori presenti nel veicolo

Tabella 6.2: Tabella contenente frasi di riferimento con variazioni testate ed effetto ottenuto

Per quanto concerne invece le variazioni di struttura sono state testate due per ogni frase che l'utente è previsto dire all'assistente nelle conversazioni di ogni specifico caso d'uso. In particolare, nei casi in cui fosse possibile, sono state testate due variazioni per frase per ogni accessorio. Ad esempio per le richieste di informazioni sono state testate le frasi della Tabella 6.2 sostituendo ogni volta l'oggetto con il nome di un altro accessorio presente nel database. Invece, per i comandi, è sempre stata utilizzata la struttura della frase come mostrato nella Tabella 6.2, ma sostituendo solamente il nome dell'oggetto con quello di un altro accessorio in grado di supportare tale operazione (ad esempio l'azione di accensione per i fendinebbia). L'assistente ha reagito correttamente anche nel caso venisse richiesto un accessorio non supportato. Infatti se la richiesta si riferisce ad un accessorio non conosciuto, l'assistente risponde con una frase di default per segnalare all'utente che non possiede informazioni riguardanti l'accessorio richiesto. L'assistente infine non supporta comandi del tipo "Tira il freno a mano" o altre espressioni gergali. Le conversazioni complete testate sono state omesse per brevità e per evitare ridondanza con le frasi già presentate nella Tabella 6.2.

Al termine di questa fase l'assistente ha dimostrato di riuscire a portare a termine il 90% delle conversazioni testate, dove sono state utilizzate le frasi e i vocaboli d'esempio della Tabella 6.1 e della Tabella 6.2. Gli unici casi in cui non è riuscito a portare a termine la conversazione sono stati quelli nei quali, al posto del nome dell'accessorio, è stata utilizzata l'azione effettuata da esso. Ha dimostrato problemi, ad esempio, nell'associazione della parola radio con musica, chiedendo di ripetere il comando dato che la parola "musica" non veniva associata a nessun accessorio. Di conseguenza, al fine di portare a termine correttamente la conversazione, è stato necessario ripetere il comando come nella situazione ideale dei casi d'uso presentati al Capitolo 3.

Per potere aggiungere questa possibilità di correlare l'azione all'accessorio relativo è stato necessario introdurre, in una fase successiva, l'esplicita associazione tra i due in DialogFlow.

Terza fase: cambio di contesto

Nell'ultima fase è stato testato il corretto comportamento dell'assistente se sottoposto ad un cambio di contesto nel mezzo della conversazione.

Per quest'ultima fase è stato definito come successo il corretto completamento della conversazione per l'**ultimo** caso d'uso avviato. Ad esempio:

- viene iniziata una conversazione come definita per il caso d'uso descritto nella Sezione 3.3.1;
- a metà conversazione si cambia al caso d'uso mostrato nella Sezione 3.3.2;

- nel mezzo della conversazione viene cambiato di nuovo contesto al caso d’uso della Sezione 3.3.4;
- si conclude la conversazione per quest’ultimo caso d’uso.

Il successo è verificato se l’assistente riesce a concludere correttamente la conversazione come definita per l’ultimo caso d’uso attivato.

Sono state testate due situazioni, simulabili utilizzando i casi d’uso stabiliti:

1. l’utente varia il contesto della conversazione: la conversazione viene guidata dall’utente che varia tra i casi d’uso delle categorie *Comando*, *Richiesta di informazioni* e *Richiesta di informazioni derivate*;
2. il cambio di contesto è forzato dall’assistente: a conversazione iniziata viene attivato un evento che causa una situazione di *Alert critico* o *Avviso*, il contesto quindi viene spostato per la gestione del possibile pericolo.

Caso d’uso iniziale	Caso d’uso inserito durante la conversazione	Caso d’uso portato a termine
Comando	Richiesta di informazioni	Richiesta di informazioni
Richiesta di informazioni (derivate)	Comando	Comando
Richiesta di informazioni	Comando	Comando
Comando	Avvisi	Avvisi
Richiesta di informazioni	Alert critico	Alert critico
Alert critico	Richiesta di informazioni	Richiesta di informazioni

Tabella 6.3: Tabella le combinazioni di casi d’uso testate (singola variazione).

È stato testato il passaggio tra due contesti arbitrari tra: *Comando*, *Richiesta di informazioni*, *Richiesta di informazioni (derivate)*, *Avvisi* e *Alert critici*. Tali cambi

di contesto sono mostrati nella Tabella 6.3. Questa verifica è stata effettuata avviando innanzitutto la conversazione come previsto per il tipo di caso d’uso indicato nella prima colonna della Tabella 6.3 e nel mezzo della stessa è stato cambiato contesto a quello della conversazione prevista per il caso d’uso indicato nella seconda colonna della Tabella 6.3. I test hanno incluso anche casi in cui la conversazione è iniziata dall’assistente stesso sia come prima conversazione avviata che come conversazione alla quale si cambia nel mezzo del discorso. Sono stati testati in totale sei possibilità: in quattro situazioni è l’utente a cambiare contesto di conversazione, nelle altre due è l’assistente a effettuare il cambio di discorso.

È stato testato inoltre un doppio cambio di contesto. Le combinazioni tentate in questo caso, sono disponibili nella Tabella 6.4. Il processo è identico al precedente, ma ripetuto due volte prima di portare a termine la conversazione. Ad esempio è stato testato il passaggio da *Richiesta di informazioni (derivate)* a un *Comando* e infine a un contesto di *Richiesta di informazioni*.

Nonostante l’assistente sia stato in grado di seguire correttamente il 100% degli

Caso d’uso iniziale	Prima variazione della conversazione	Seconda variazione della conversazione	Caso d’uso portato a termine
Richiesta di informazioni (derivate)	Comando	Richiesta di informazioni	Richiesta di informazioni
Comando	Richiesta di informazioni	Comando	Comando
Comando	Richiesta di informazioni	Avvisi	Avvisi
Richiesta di informazioni	Alert critico	Comando	Comando

Tabella 6.4: Tabella le combinazioni di casi d’uso testate (doppia variazione).

incroci tentati, portando a termine le conversazioni per l’ultimo contesto attivato, non è possibile affermare che sia efficace in tutte le situazioni poichè impedisce all’utente di ignorare la conversazione se iniziata dall’assistente. Questo limite è emerso quando il contesto della conversazione viene cambiato dall’assistente stesso. In questi casi viene totalmente abbandonata la conversazione precedente. Di conseguenza se viene ignorata la conversazione iniziata dal veicolo, non è possibile tornare a quella antecedente. A questo scopo sono stati fatti test ulteriori: sono stati verificati singoli cambi di contesto indotti alla conversazione da parte dell’assistente. Le variazioni provate sono visibili nella Tabella 6.5. In questa ultima verifica è stata testata la possibilità di ritornare al discorso precedente l’inizio della conversazione avviata dall’assistente.

Questo ultimo test ha portato alla luce il limite dell’assistente di non potere recu-

Caso d'uso iniziale	Caso d'uso avviato dall'assistente
Comandi	Avvisi
Richiesta di informazioni	Avvisi
Richiesta di informazioni (derivate)	Alert critico

Tabella 6.5: Tabella le combinazioni di casi d'uso testate (cambio contesto indotto dall'assistente).

perare il discorso precedente all'avvio della conversazione per Avvisi o Alert critici. Infatti sono state testate tre combinazioni, una per ogni possibile conversazione che l'utente può avviare e l'assistente non è riuscito in nessuno dei casi a ritornare al discorso antecedente il suo avvio della conversazione. In conclusione, nel caso in cui l'utente ignori l'alert non è possibile recuperare il vecchio contesto a meno che non si resettì la conversazione.

Questo limite può essere oggetto di lavori futuri e potrebbe essere risolto, per esempio, mantenendo più contesti di conversazione in memoria allo stesso tempo e attribuendo all'ultimo attivato la priorità più alta, di modo che esso sia il primo considerato come prosecuzione naturale della conversazione quando una nuova frase viene ricevuta in input. Il numero di contesti da tenere in memoria però può variare, per questo è necessario uno studio utente per capire i bisogni di un guidatore e verificare se è sufficiente tenere solo il penultimo e l'ultimo contesto attivati oppure è necessario tenere traccia di un maggior numero di essi.

Nella Figura 6.4 è rappresentata una conversazione dove il contesto viene cambiato

```
[Client] Utente: Accendi i tergicristalli
> [Client] Assistente: Quali?
[Client] Utente: come si accendono i fendinebbia
> [Client] Assistente: Quali?
Assistente: L'indicatore pressione pneumatici ha cambiato stato e ora si trova nello stato acceso ←
[Client] Utente: Che cosa significa
> [Client] Assistente: questa spia viene accesa in caso la pressione degli pneumatici sia bassa, oppure se sono state cambiate le gomme di recente. Hai cambiato le gomme di recente? ←
[Client] Utente: anteriori
> [Client] Assistente: Non penso di seguirti. Potresti ripetere?
[Client] Utente: sì ←
> [Client] Assistente: Perfetto, allora spengo la spia. ←
```

Figura 6.4: Figura contenente un esempio di conversazione con cambio contesto.

due volte. una dal guidatore e una dall'assistente, portando alla luce l'impossibilità di tornare al discorso antecedente la conversazione avviata dall'assistente. La prima volta viene cambiato dopo avere richiesto di accendere i tergicristalli, l'assistente richiede quali e il guidatore ribatte chiedendo le istruzioni per accendere i fendinebbia.

Anche in questo caso tale accessorio è presente sia anteriormente che posteriormente, di conseguenza l'assistente chiede maggiori informazioni sulla posizione. È in questo momento che viene attivata la spia della pressione degli pneumatici al fine di attivare l'avviso da parte dell'assistente. L'utente richiede più informazioni a proposito di tale segnale. L'assistente risponde con il significato di tale spia. A questo punto l'utente fornisce informazioni sulla posizione dei fari, per rispondere alla domanda lasciata in sospeso prima, ma il contesto della conversazione precedente è stato cancellato essendo che l'assistente ha attivato un *avviso*, per cui non è possibile proseguire con quella conversazione. Tale richiesta risulta incomprensibile all'assistente che avvisa il guidatore di non avere capito le sue intenzioni. Proseguendo la conversazione come prevista per il caso d'uso *Avviso*, essa viene conclusa con successo. Infatti la risposta "Sì", non è interpretata dall'assistente come risposta alla domanda "Potresti ripetere?", bensì come risposta alla domanda dell'assistente lasciata in sospeso precedentemente quando è stato riferito il significato della spia appena accesa. Ciò porta alla conclusione della conversazione come prevista per il caso d'uso in una situazione ideale.

Nella Figura 6.4 sono evidenziate con a fianco una freccia rossa, tutte le frasi facenti parte del caso d'uso *Avviso*. Si può notare come la conversazione sia correttamente portata a termine come nel caso ideale essendo l'ultima avviata e la frase "Anteriori" è gestita come un errore introdotto durante il discorso.

Infine nelle Figure 6.5 e 6.6 sono rappresentati due esempi quando vengono introdotti errori nel mezzo della conversazione.

Nella Figura 6.5 è rappresentato il caso in cui l'assistente inizia una conversazione. Ci troviamo nella situazione di un *avviso* da parte del veicolo che la spia della pressione degli pneumatici è accesa. Il guidatore pronuncia direttamente una frase anomala. L'assistente afferma di non avere recepito correttamente il messaggio e chiede all'utente di ripetere la frase. A questo punto il guidatore richiede all'assistente spiegazioni in più sul significato della spia in questione. L'assistente non perde il contesto della conversazione, portandola a termine con successo.

Nella Figura 6.6 è presentato il caso in cui la conversazione è avviata dal guidatore per richiedere l'accensione dei fendinebbia. Nel mezzo viene pronunciata una frase anomala quando l'assistente richiede informazioni in più sulla posizione dei fari interessati. Dopo che l'assistente chiede di ripetere la frase, l'utente fornisce tali informazioni, permettendo, di fatto, all'assistente di portare a termine la conversazione correttamente eseguendo il comando richiesto.

6.3 Latenza di risposta dell'assistente

Al fine di testare la reattività dell'assistente, sono stati misurati i tempi di risposta in diversi casi d'uso. La velocità della connessione utilizzata è visualizzata nella Figura 6.7.

```
Assistente: L'indicatore pressione pneumatici ha cambiato stato e ora si trova nello stato acceso
[Client] Utente: bla bla bla
> [Client] Assistente: Non ti ho sentito bene. Puoi ripetere?
[Client] Utente: Che cosa significa
> [Client] Assistente: questa spia viene accesa in caso la pressione degli pneumatici sia bassa, oppure se sono state cambiate le gomme di recente. Hai cambiato le gomme di recente?
[Client] Utente: bla bla
> [Client] Assistente: Potresti dirlo ancora una volta?
[Client] Utente: si
> [Client] Assistente: Perfetto, allora spengo la spia.
```

Figura 6.5: Figura contenente un esempio di errore durante la conversazione iniziata dall'assistente.

```
[Client] Utente: Accendi fendinebbia
> [Client] Assistente: Quali?
[Client] Utente: Andreoli
> [Client] Assistente: Non penso di seguirti. Potresti ripetere?
[Client] Utente: posteriori
> [Client] Assistente: Azione effettuata.
```

Figura 6.6: Figura contenente un esempio di errore durante la conversazione iniziata dal guidatore.

Per calcolare la latenza tra la richiesta dell'utente e la risposta dell'assistente è stato misurato l'intervallo di tempo che intercorre tra la lettura dell'input dalla DataLine del microfono e la ricezione effettiva di una risposta da presentare all'utente. Il tutto richiede diversi passaggi:

1. lettura dell'input;
2. traduzione da vocale a testo;
3. invio a DialogFlow per l'estrazione dei dati di interesse;
4. elaborazione di un responso da parte della logica dell'applicativo (può richiedere interazioni con il database);
5. invio del Fulfilment a DialogFlow;
6. ricezione e estrazione della risposta dal response ricevuto da DialogFlow.

In base alle misure ottenute con le caratteristiche di connessione di rete della Figura 6.7 è risultato che:

- il **tempo massimo** di risposta è risultato *1003 ms*;
- il **tempo minimo** di risposta è risultato *360 ms*;
- il **tempo medio** di risposta è risultato *688.85 ms*.

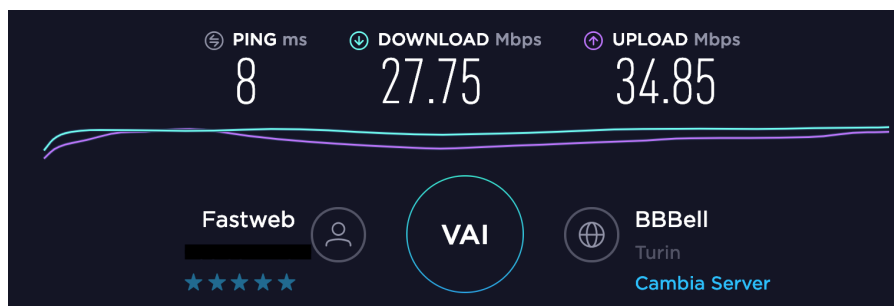


Figura 6.7: Velocità della connessione di rete usata durante la misurazione delle latenze.

Questi risultati sono frutto anche della velocità della scheda di rete dell'elaboratore sul quale è stata impiegata l'applicazione e della connessione usata, in un contesto di veicoli connessi, in costante movimento, probabilmente le cifre possono aumentare. Ad ogni modo tali numeri dimostrano che l'utilizzo di cloud service al fine di fornire la possibilità di un'interazione vocale con il veicolo non introduce latenze così elevate da impedire una conversazione. Questo indica la possibilità di un'apertura più estensiva all'utilizzo di servizi cloud per veicoli connessi, snellendo le applicazioni impiegate sulle centraline delle automobili.

6.4 Considerazioni finali

6.4.1 Limite nella gestione dei contesti

L'assistente presenta alcuni limiti nella gestione dei contesti delle conversazioni. Se viene seguita una sequenza precisa di operazioni, l'assistente funziona correttamente. Nel caso in cui sia ignorato il filo del discorso proposto dal veicolo ha delle difficoltà. In altre parole, nei cambi di contesto:

- se il contesto della conversazione viene cambiato dall'utente, quindi è il guidatore a comandare la conversazione, non sussiste alcun problema;
- quando, invece, il contesto della conversazione viene cambiato dall'assistente, ad esempio per effettuare un avviso o un alert critico, essendo che, come detto in precedenza, l'assistente è stato pensato per seguire l'ultimo contesto di conversazione, il contesto precedente non viene salvato e quindi se l'utente ignora l'alert, ma vuole continuare la conversazione già iniziata, dovrà riprenderla dall'inizio.

6.4.2 Interazione con il guidatore

L'assistente sviluppato in questa tesi non è da considerarsi come una sostituzione per quelli attualmente presenti in commercio, bensì come una semplificazione di essi. Infatti nonostante le soluzioni commerciali migliorino sicuramente l'esperienza di guida a bordo di un veicolo, non la semplificano. Il prototipo che è stato sviluppato ha snellito l'interazione con il guidatore al minimo indispensabile: la voce, la visualizzazione di dati sul cruscotto e l'attivazione dell'ascolto da parte dell'assistente tramite un bottone. Non sono stati introdotti touch screens, menù e leve aggiuntivi, è stato impiegato solo il minimo indispensabile. Purtroppo, però, un modo per segnalare la necessità di attivazione dell'ascolto (bottone) è stato necessario, di modo da evitare che l'assistente reagisse a conversazioni all'interno dell'abitacolo. Tale bottone nell'implementazione attuale corrisponde alla barra spaziatrice dell'elaboratore sul quale viene impiegata l'applicazione centrale (modulo Intelligence), ma se trasportato a bordo di un veicolo può essere adattato in una posizione più comoda per il guidatore, ad esempio sul volante. Da questo punto di vista è possibile migliorare l'interazione utilizzando, per l'attivazione dell'ascolto, parole chiave.

Infine il fatto che l'assistente sia in grado di avvisare il guidatore in caso di problemi più o meno gravi all'interno del motore può risultare efficace soprattutto in quei casi in cui l'utente non conosce il significato degli indicatori mostrati sul cruscotto. Per verificare questo però è necessario uno studio utente, che in questa tesi non è stato affrontato, come espresso nel capitolo 7.

Capitolo 7

Conclusioni

L'obiettivo di questa tesi è quello di esplorare nuove possibilità di interazione tra guidatore e veicolo. A tale scopo è stato sviluppato un assistente conversazionale che cambiasse rotta rispetto ai tradizionali assistenti presenti sul mercato, senza però rivoluzionare totalmente le sue modalità di utilizzo. Come per gli assistenti già presenti sui veicoli attuali (Seat, BMW, Mercedes) l'interazione con l'applicativo è di tipo vocale e visuale, ma al contrario delle soluzioni in commercio l'assistente sviluppato prevede una forte integrazione con il veicolo e la proprietà di essere proattivo. Una soluzione reattiva consiste in un assistente che è attivabile solamente dall'utente e quindi la conversazione con il guidatore risulta sempre iniziata da quest'ultimo. La soluzione proattiva invece prevede che sia anche l'assistente a potere iniziare la conversazione con il guidatore. Questa caratteristica ha aggiunto la possibilità di potere notificare l'utente di eventuali problemi al motore dei quali non si sarebbe potuto accorgere e fornire anche addizionali informazioni su come risolverli.

L'assistente sviluppato si è dimostrato funzionante per i casi d'uso particolari utilizzati come test funzionali, in particolari tali casi d'uso puntano a coprire situazioni in cui vi è una richiesta/comando da parte di un utente o vi è necessità di avvisare il guidatore di un cambio di stato all'interno del veicolo (semplice avviso oppure alert critico).

Al fine di raggiungere il risultato atteso, la creazione dell'assistente è stata approssiata per gradi:

- analisi delle soluzioni in commercio;
- definizione dei casi d'uso per il testing;
- definizione dell'architettura del sistema;
- decisione delle tecnologie da utilizzare;
- sviluppo;

- testing funzionale rispetto ai casi d’uso definiti.

La prima fase ha richiesto la ricerca delle soluzioni attualmente sul mercato e l’analisi dei pro e contro di ciascuna di esse. In particolare sono state analizzate le soluzioni di Seat, BMW e Mercedes. Il minimo comune denominatore tra di esse è quello di essere soluzioni reattive e non a conoscenza del reale stato del veicolo, che si traduce in pratica nell’incapacità dell’assistente di notificare da sé il guidatore di eventuali problemi all’interno del motore.

Al fine di ovviare a questi due “difetti” trovati negli assistenti tradizionali, nella fase successiva, sono stati definiti dei casi d’uso per testare le effettive proprietà dell’assistente sviluppato. Dopo ciò il passaggio successivo ha previsto la definizione dell’architettura del sistema: il prototipo è stato concepito come applicazione distribuita, l’intelligenza principale risiede all’interno del veicolo ed è responsabile della logica interna e dell’interazione con il guidatore e sfrutta cloud services per la modellazione delle conversazioni e traduzione da vocale a testuale e viceversa. Le principali tecnologie utilizzate per la realizzazione dell’assistente sono state: Assetto Corsa (simulatore dal quale ottenere dati telemetrici), DialogFlow (per la strutturazione delle conversazioni) e alcuni cloud services forniti da Google (per la traduzione text-to-speech e speech-to-text). Durante la fase di sviluppo, essendo che il cruscotto presente all’interno di Assetto Corsa non conteneva informazioni sugli accessori a bordo del veicolo, è stata introdotta una dashboard integrativa al fine di offrire una visualizzazione sia dei dati telemetrici che dello stato delle spie degli accessori nell’auto. In aggiunta a ciò tale cruscotto permette il cambiamento dello stato degli accessori, utile al fine di testare il corretto comportamento dell’assistente.

Nell’ultima fase è stato verificato che l’assistente assumesse il comportamento aspettato rispetto ai casi d’uso definiti all’inizio. In particolare l’assistente regisce correttamente a tutte le situazioni dei casi d’uso e inoltre è in grado di adattarsi correttamente nel caso di cambi di contesto, ad esempio: nel caso in cui la conversazione sia stata iniziata nel contesto del caso d’uso 1, ma nel mezzo del dialogo si cambia al contesto di caso d’uso 2, l’assistente è in grado di proseguire la conversazione per il caso d’uso 2.

7.1 Lavori futuri

In conclusione l’idea del progetto è stimolante e nuova, soprattutto per il fatto che questo tipo di assistente si pone l’obiettivo di avere una forte integrazione con il veicolo, cosa che i classici assistenti non hanno. Ci sono però ancora diversi spunti di miglioramento:

- l’assistente presente poca varietà nella struttura dei responsi presentati alle richieste dell’utente. A questo scopo si potrebbero aggiungere degli algoritmi

di intelligenza artificiale per fare variare la tali risposte quando ritornate più volte;

- non è stato eseguito un reale studio utente. L'assistente è stato solamente testato funzionante per i casi d'uso definiti, ma un'eventuale studio utente potrebbe risultare utile per portare alla luce miglie da apportare all'interazione guidatore-veicolo.

Bibliografia

- [1] C. Magazine, “What is a conversational ui and why does it matter.” [Online]. Available: <https://chatbotsmagazine.com/what-is-a-conversational-ui-and-why-it-matters-de358507b9a2>
- [2] D. R. Large, L. Clark, A. Quandt, G. Burnett, and L. Skrypchuk, “Steering the conversation: A linguistic exploration of natural language interactions with a digital assistant during simulated driving,” *Applied Ergonomics*, vol. 63, pp. 53 – 61, 2017. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0003687017300790>
- [3] C. SRI International, Menlo Park, “Vehicle personal assistant.” [Online]. Available: <https://patentimages.storage.googleapis.com/75/6e/0d/f956a0539fbb74/US9085303.pdf>
- [4] Blogio, “Seat lancia amazon alexa in italia.” [Online]. Available: <http://www.autoblog.it/post/931944/seat-lancia-amazon-alexa-in-italia>
- [5] C. Magazine, “Amazon alexa tested in a seat ateca.” [Online]. Available: <https://www.carmagazine.co.uk/car-news/we-test-amazon-alexa-in-a-seat-ateca-five-things-we-learned/>
- [6] Autoappassionati.it, “Seat lancia l’assistente vocale amazon alexa anche in italia.” [Online]. Available: <https://www.autoappassionati.it/seat-amazon-alexa-italia/>
- [7] motor1.com, “Mercedes mbux, la rivoluzione nel cruscotto.” [Online]. Available: <https://it.motor1.com/news/226876/mercedes-mbux-al-ces-2018-la-stella-presenta-il-nuovo-super-cruscotto/>
- [8] C. dello Sport, “Mercedes classe a, siamo saliti a bordo in anteprima al ces di las vegas.” [Online]. Available: http://www.corrieredellosport.it/news/motori/saloni/2018/01/10-36761859/mercedes_classe_a_siamo_saliti_a_bordo_in_anteprema_al_ces_di_las_vegas/
- [9] W. News, “Mercedes classe a: sistema d’infotainment mbux.” [Online]. Available: <https://www.webnews.it/2018/10/15/mercedes-classe-a-sistema-infotainment-mbux/>

- [10] Blogo, “Hey bmw: l’intelligent personal assistant debutta nel 2019.” [Online]. Available: <http://www.autoblog.it/post/921880/hey-bmw-lintelligent-personal-assistant-debutta-nel-2019>
- [11] Autoappassionati.it, “Hey bmw, il nuovo assistente vocale della casa tedesca.” [Online]. Available: <https://www.autoappassionati.it/hey-bmw-assistente-vocale/>
- [12] bradland, “Ac telemetry - an assetto corsa telemetry logger.” [Online]. Available: https://github.com/bradland/ac_telemetry
- [13] ottonello, “Assettocorsatelemetry.” [Online]. Available: <https://github.com/ottonello/AssettoCorsaTelemetry>
- [14] joaoubaldo, “Acudpclient.” [Online]. Available: <https://github.com/joaoubaldo/acudpclient>
- [15] K. Simulazioni, “Ac socket documentation.” [Online]. Available: https://docs.google.com/spreadsheets/d/1UTgeE7vbnGIZDz-URRk2eBIPc_LR1vWcZklp7xD9N0Y/