Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Riconoscimento del linguaggio naturale per assistenti virtuali: interpretazione e apprendimento di comandi vocali

Tesi di laurea triennale

Relatore

Prof.Paolo Baldan

Laureando

Massimo Toffoletto

Anno Accademico 2019-2020

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 320 ore, dal laureando Massimo Toffoletto presso l'azienda Zucchetti S.p.A. Gli obiettivi da raggiungere sono stati molteplici e suddivisi in due parti.
Nella prima parte ho studiato il funzionamento dei tre principali assistenti virtuali nel seguente ordine: Assistant, Alexa e Siri. Gli obiettivi sono stati quindi l'analisi dei singoli assistenti e lo svolgimento di una comparazione tra gli stessi, sia delle funzionalità offerte agli sviluppatori che delle capacità riconoscitive del linguaggio naturale. A questo proposito ho realizzato un proof of concept dimostrativo per ciascuno di essi.
Nella seconda parte gli obiettivi sono stati lo studio di regole per la realizzazione di grammatiche che interpretano il linguaggio naturale, implementate da Zucchetti ma ancora in via di sviluppo, e la costruzione di un'applicazione che le utilizzi. Infine ho implementato un'interfaccia vocale con capacità conversazionale di interagire che si integri con la tecnologia Zucchetti.

"Il computer non è una macchina intelligente che aiuta le persone stupide, anzi, è una macchina stupida che funziona solo nelle mani delle persone intelligenti."

— Umberto Eco

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Paolo Baldan, relatore della mia tesi, per l'aiuto e il sostegno che mi ha fornito durante lo svolgimento del lavoro.

Desidero ringraziare con affetto tutta la mia famiglia per il loro sostegno e per essere stati sempre presenti durante il mio percorso di studi.

Voglio inoltre ringraziare tutti i miei amici per gli anni meravigliosi trascorsi assieme ed in particolar modo Alberto, Martina, Lorenzo e Fiammetta che mi hanno sempre sostenuto nei momenti più difficili ma anche in quelli più felici.

Padova, Luglio 2020 Massimo Toffoletto

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Descrizione del progetto

Il progetto di stage è legato ad uno dei rami più conosciuti dell'intelligenza artificiale: l'interpretazione del linguaggio naturale. Questo argomento non è trattato nel percorso di studi della laurea triennale e personalmente lo ritengo una sfida ed una motivazione aggiuntiva per conoscere e scoprire nuove tecnologie.
Nelle prime settimane il lavoro si è articolato in attività di ricerca, sperimentazione e documentazione su tre assistenti virtuali: Assistant, Alexa e Siri. Successivamente ho studiato la tecnologia Zucchetti per l'interpretazione del linguaggio naturale e ho realizzato un'applicazione basata su di essa, che rispetti i principi di progettazione appresi durante le precedenti ricerche ed implementi un meccanismo di conversazione.

1.2 Zucchetti S.p.A

Il progetto di stage è stato svolto con Zucchetti, un'azienda italiana fondata più di 40 anni fa che produce soluzioni software, hardware e servizi per soddisfare le esigenze tecnologiche dei propri clienti, anche a livello internazionale. Le sedi sono dislocate in numerose città italiane tra cui Padova dove ho svolto lo stage e Lodi che rappresenta la sede amministrativa.

Figura 1.1: Logo di Zucchetti

Domenico Zucchetti, fondatore dell'azienda, ha avuto la geniale intuizione di costruire un software per agevolare il lavoro dei commercialisti, allora completamente cartaceo e manuale. Con il passare degli anni il suo prodotto ha riscontrato sempre maggior successo tanto da ottenere collaborazioni con aziende del calibro di *IBM*[g]. A partire

da questo, Zucchetti ha continuato a perseguire la strada dell'innovazione integrando il proprio prodotto con moduli nuovi, quali *ERP*[g]e la più recente fatturazione elettronica, per conferire maggiore flessibilità e adattabilità ad ogni tipologia di impresa, senza limitarsi ai commercialisti.
Forte del suo prodotto, negli ultimi decenni l'azienda si è espansa a livello nazionale ed internazionale, ponendosi sul mercato con una vasta gamma di servizi per innumerevoli settori tra cui industria manifatturiera, trasporti, logistica, sanità, fitness e molti altri.
Uno dei pilastri di Zucchetti che ha indubbiamente contribuito alla sua espansione è l'innovazione e la propensione alla ricerca di nuove tecnologie. A questo proposito la sede di Padova è composta da un reparto di ricerca e sviluppo, nel quale sono stato inserito durante la mia esperienza di stage, coordinato dal dott. Gregorio Piccoli che mi ha proposto il progetto e si è offerto come mio tutor.

1.3 Lo stage proposto

1.3.1 Contesto del lavoro

L'azienda sta cercando di introdurre nei propri prodotti, in particolare nel software gestionale, un'interfaccia vocale che permetta agli utenti di interagire in modo più veloce e spontaneo nelle operazioni comuni. Il loro obiettivo è quindi implementare un'interfaccia diversa da quella grafica, con caratteristiche proprie, che esprima un modo nuovo di comunicare con le applicazioni. Esso, seppur ancora poco sviluppato, possiede grandi potenzialità.
Per raggiungere questo traguardo il principale ostacolo da superare è il riconoscimento del linguaggio naturale attraverso un'applicativo software. Perciò l'azienda ha sviluppato delle regole per la creazione di *grammatiche* che permettano di eseguire il *parsing* dei comandi vocali. Tuttavia è una tecnologia ancora in fase di sviluppo e proprio per questo, come progetto di stage, mi sono state proposte due tematiche mirate ad un'attività di esplorazione. Esse sono pensate per suddividere il lavoro in due parti:

1. analizzare i tre assistenti virtuali attualmente più diffusi sul mercato, Assistant, Alexa e Siri, per comprenderne le abilità e verificare i loro possibili impieghi nei prodotti dell'azienda;

2. implementare una *NLU*[g]basata su una *grammatica*[g]costruita con la nuova tecnologia Zucchetti, possibilmente con capacità conversazionale, ed ispirata agli assistenti virtuali precedentemente analizzati.

Più nello specifico gli obiettivi da perseguire sono descritti nella sezione 2.2 ma possono essere riassunti nei seguenti punti:

* analizzare le tecnologie offerte ed i principi di implementazione delle abilità di Assistant, Alexa e Siri con *PoC*[g]che le concretizzano;

* capire dalle ricerche svolte se esistono componenti rilevanti per i progetti aziendali che riguardano la realizzazione di un'interfaccia vocale;

* implementare un'applicazione ispirata alle nozioni studiate e che utilizzi una *grammatica* costruita mediante la tecnologia Zucchetti, possibilmente con capacità conversazionale.

1.3.2 Problematiche incontrate

Lo stage è stato svolto esclusivamente da remoto e ciò ha in parte influenzato il piano di lavoro, ponendo come facoltativi degli obiettivi quasi sicuramente raggiungibili in presenza come quelli dell'attività legata a Siri. Infatti, per la maggior parte dei compiti, necessita di un computer con sistema operativo MacOS che ho potuto reperire solo per un breve periodo di tempo ed inoltre, a causa delle restrizioni dell'account sviluppatore Apple a disposizione, non si è potuta concludere l'attività.
Ad ogni modo, considerando la nuova esperienza di lavoro remoto sia per me che per l'azienda, lo stage è stato molto positivo e non sono emerse ulteriori problematiche.

1.3.3 Sintesi dei risultati

Nonostante il problema legato all'attività con Siri, è stato possibile concludere lo stage rispettando il piano di lavoro e con risultati ottimi.
Gli esiti della ricerca delle prime settimane hanno soddisfatto le aspettative, confermando alcune intuizioni del tutor e facendo emergere nuovi elementi importanti per gli sviluppi della loro interfaccia vocale. Le conferme riguardano l'utilizzo di Assistant ed Alexa per comunicare con i prodotti aziendali attraverso appositi *SDK*[g] offerti rispettivamente da Google e Amazon; Apple invece non fornisce lo stesso per Siri affermando la sua politica di ecosistema chiuso. I nuovi elementi rilevati sono le tecniche di progettazione e realizzazione della capacità conversazionale e l'integrazione dell'assistente virtuale all'interno di applicazioni mobile, fornita solo da Assistant e Siri. Essi sono emersi principalmente grazie ai *PoC* costruiti in modo mirato sulle funzionalità più interessanti per l'azienda e hanno fornito molti spunti di riflessione su possibili nuove implementazioni. I loro risultati sono così riassunti:

* *PoC* di Assistant: utilizzo dell'assistente virtuale per attivare determinate funzionalità di un'applicazione. Nel mio caso si tratta di un timer per fare attività fisica;

* *PoC* di Alexa: effettuare una conversazione con l'assistente virtuale finalizzata ad eseguire una determinata funzione. Nel mio caso si tratta di riferire la data di nascita al mio applicativo;

* *PoC* di Siri: utilizzo dell'assistente virtuale per creare comandi personalizzati di attivazione delle applicazioni o delle loro specifiche funzionalità. Sebbene non sia stato possibile completarlo con successo, i dati ricavati sono stati ugualmente soddisfacenti.

Anche l'esito dell'applicazione ha dato ottimi frutti, confermando le caratteristiche delle *grammatiche* Zucchetti ovvero facilità e grande flessibilità nell'utilizzo e portando alla luce nozioni e tematiche rilevanti per gli sviluppi della loro *NLU*, quali capacità di memorizzazione, comprensione del contesto e architettura del software che ne fa uso. L'applicazione è costituita da una *NLU* capace di interpretare data di nascita e di compleanno attraverso una conversazione. Questo significa, ad esempio, che l'utente può fornire una data parziale ed in seguito l'applicazione chiederà i dati mancanti con domande mirate.

1.4 Organizzazione del testo

Il capitolo finora trattato è di introduzione mentre il seguito del documento è organizzato secondo questa struttura:

Il secondo capitolo descrive obiettivi, pianificazione, strumenti e tecnologie del progetto di stage ed infine le motivazioni che mi hanno portato a sceglierlo.

Il terzo capitolo descrive il lavoro di analisi e ricerca sugli assistenti virtuali Assistant, Alexa e Siri e le considerazioni tratte al termine.

Il quarto capitolo descrive inizialmente la tecnologia Zucchetti ed in seguito l'applicazione costruita per la comprensione del linguaggio naturale con capacità conversazionale.

Il quinto capitolo rappresenta le conclusioni dell'elaborato: riassume il lavoro svolto, il raggiungimento degli obiettivi ed infine riporta un'analisi retrospettiva dell'intera esperienza di stage.

Riguardo la stesura del testo di questo documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Lo stage

2.1 Descrizione del progetto

Il progetto di stage è legato ad uno degli ambiti più innovativi della ricerca scientifica e informatica: il riconoscimento e l'elaborazione del linguaggio naturale da parte di un calcolatore.
L'azienda sta affrontando la sfida della comprensione di comandi vocali per permettere agli utenti di comunicare a voce con i propri prodotti. Infatti stanno sviluppando una loro tecnologia caratterizzata da regole deterministiche capaci di generare grammatiche che riconoscono il linguaggio naturale.
Su questo argomento è stato costruito il progetto di stage che si articola in due macro parti:

1. studio degli assistenti virtuali più comuni ed utilizzati;
2. creazione di un'applicazione sotto forma di *PoC* in grado di intrattenere una conversazione.

La prima parte richiede un'analisi dettagliata e comparativa dei tre assistenti virtuali più utilizzati:

* Assistant: assistente virtuale di Google;
* Alexa: assistente virtuale di Amazon;
* Siri: assistente virtuale di Apple.

Lo studio deve comprendere le loro capacità conversazionali e di interazione con le applicazioni, prestando particolare attenzione alla costruzione di abilità personalizzate e integrate in prodotti che comunichino con gli assistenti ma posseggano una propria *NLU*. Le altre capacità suscitano minor interesse in quanto l'azienda punta alla realizzazione di un assistente utile solo ad interfacciarsi con i propri prodotti e non su larga scala. Inoltre, per dare concretezza alle funzionalità riscontrate e più significative, è richiesto un *PoC* per ogni assistente.
La seconda parte del lavoro consiste nella creazione di un'applicazione con una propria *NLU* basata su grammatiche costruite mediante la tecnologia Zucchetti ed un'interfaccia vocale che permetta di intrattenere una conversazione con gli utenti.
Il contenuto dell'applicazione non è vincolato dall'azienda in quanto l'obiettivo è approfondire le capacità della loro tecnologia e non costruire un'applicazione direttamente

utilizzabile nei software Zucchetti. In comune accordo è stato scelto il contesto della data di nascita poiché ricco di sfumature sia nelle richieste che nelle risposte e possiede parametri specifici e funzionali alla sperimentazione della capacità di conversazione. Per concludere, lo scopo generale dello di stage è svolgere un'attività di ricerca e di approfondimento nella tematica del linguaggio naturale su tecnologie già presenti sul mercato e su quella Zucchetti.

2.2 Obiettivi

2.2.1 Classificazione

La classificazione degli obiettivi permette di identificarli univocamente adottando la seguente notazione:

* *OB-x*: obiettivi obbligatori, vincolanti e primari richiesti dall'azienda;
* *OD-x*: obiettivi desiderabili, non vincolanti o strettamente necessari ma dal riconoscibile valore aggiunto;
* *OF-x*: obiettivi facoltativi dal valore aggiunto tangibile ma difficilmente raggiungibili a causa di agenti esterni all'attività di stage.

Il simbolo 'x' rappresenta un numero progressivo intero maggiore di zero.

2.2.2 Obiettivi obbligatori

* **OB-1**: studio e analisi delle capacità e delle modalità di utilizzo lato sviluppatore di Assistant;
* **OB-2**: implementazione di un *PoC* che realizzi una funzionalità di Assistant accordata sulla base dei risultati della ricerca;
* **OB-3**: studio e analisi delle capacità e delle modalità di utilizzo lato sviluppatore di Alexa;
* **OB-4**: implementazione di un *PoC* che realizzi una funzionalità di Alexa accordata sulla base dei risultati della ricerca;
* **OB-5**: redazione di un documento che riporti un'analisi dettagliata e comparativa degli assistenti virtuali studiati;
* **OB-6**: studio di regole e caratteristiche della tecnologia Zucchetti per la costruzione di *grammatiche* che interpretano il linguaggio naturale;
* **OB-7**: realizzazione di un'applicazione con una propria *NLU* basata su una *grammatica* generata mediante la tecnologia Zucchetti, che interagisca con gli utenti tramite interfaccia vocale.

2.2.3 Obiettivi desiderabili

Inizialmente era stato identificato come obiettivo desiderabile l'implementazione di una componente di addestramento per l'applicazione finale. Tuttavia nel corso dello stage, date anche alcune interessanti funzionalità trovate, questo obiettivo è stato modificato nel seguente:

* **OD-1**: implementazione della capacità conversazionale tramite lo scambio di informazioni specifiche, possibilmente memorizzate, mirata a soddisfare una determinata funzionalità.

2.2.4 Obiettivi facoltativi

* **OF-1**: studio e analisi delle capacità e delle modalità di utilizzo lato sviluppatore di Siri;
* **OF-2**: implementazione di un *PoC* che realizzi una funzionalità di Siri accordata sulla base dei risultati della ricerca;

2.3 Pianificazione delle attività

Lo stage è stato svolto in modalità *smart working* con la possibilità di rimanere in comunicazione con il tutor aziendale attraverso Skype. In particolare l'orario di lavoro è stato 9:00-13:00, 14:00-18:00 e sono state stabilite sia una chiamata al termine di ogni giornata sia la compilazione di un registro delle attività giornaliero per tracciare e monitorare il lavoro svolto con l'obiettivo di ricevere *feedback* costanti dal tutor aziendale.
La pianificazione è stata svolta su un totale di 320 ore suddivise secondo quanto riportato nella tabella seguente.

Durata in ore	Date (inizio - fine)	Attività
40	04/05/2020 - 08/05/2020	Studio di Assistant e implementazione di un *PoC*.
40	11/05/2020 - 15/05/2020	Studio di Alexa e implementazione di un *PoC*.
40	18/05/2020 - 22/05/2020	Studio di Siri e implementazione di un *PoC*.
40	25/05/2020 - 29/05/2020	Test e documentazione comparativa di quanto svolto nelle settimane precedenti.
40	01/06/2020 - 05/06/2020	Apprendimento della tecnologia Zucchetti per il riconoscimento e l'elaborazione di comandi vocali.
40	08/06/2020 - 12/06/2020	Realizzazione di un'applicazione che implementi una *NLU* basata su una *grammatica* costruita mediante la tecnologia di Zucchetti.
40	15/06/2020 - 19/06/2020	Implementazione della capacità conversazionale con scambio e memorizzazione di informazioni.
40	22/06/2020 - 26/06/2020	Test e documentazione di quanto svolto nelle settimane precedenti.

Tabella 2.1: Pianificazione delle attività

La pianificazione di dettaglio di ogni settimana è stata strutturata sulla base degli obiettivi stabiliti e viene ora riportata.

Prima Settimana

Gli obiettivi prefissati sono:

* **OB-1**;
* **OB-2**.

Le attività previste riguardano l'analisi di Assistant sotto alcuni aspetti principali e sono:

* modalità di implemetazione degli intenti;
* progettazione delle abilità con attenzione all'interfaccia vocale;
* interazione degli utenti con l'assistente;
* modalità e sicurezza nel trasferimento dei dati;
* sviluppo di un *PoC* accordato sul momento per dare concretezza allo studio fatto.

Seconda Settimana

Gli obiettivi prefissati sono:

* **OB-3**;
* **OB-4**.

Le attività previste sono l'analisi di Alexa sotto alcuni aspetti principali:

* modalità di implemetazione degli intenti;
* progettazione delle abilità con attenzione all'interfaccia vocale;
* interazione degli utenti con l'assistente;
* modalità e sicurezza nel trasferimento dei dati;
* sviluppo di un *PoC* accordato sul momento per dare concretezza allo studio fatto.

Terza Settimana

Gli obiettivi prefissati sono:

* **OF-1**;
* **OF-2**.

Le attività previste sono l'analisi di Siri sotto alcuni aspetti principali:

* modalità di implemetazione per gli intenti;
* progettazione delle abilità con attenzione all'interfaccia vocale;
* interazione degli utenti con l'assistente;
* modalità e sicurezza nel trasferimento dei dati;
* sviluppo di un *PoC* accordato sul momento per dare concretezza allo studio fatto.

Quarta Settimana

L'obiettivo prefissato è:

* **OB-5**.

Le attività previste sono:

* verifica del funzionamento dei *PoC* realizzati;
* stesura della documentazione che riporta in modo dettagliato le tecnologie studiate nelle settimane precedenti.

Quinta Settimana

L'obiettivo prefissato è:

* **OB-6**.

Le attività previste sono:

* studio dell'algoritmo di Zucchetti;
* esecuzione di alcune prove concrete per verificarne il corretto apprendimento.

Sesta Settimana

L'obiettivo prefissato è:

* **OB-7**

Le attività previste sono:

* progettazione di un'applicazione, anch'essa sotto forma di *PoC*, che implementi una propria *NLU* con capacità conversazionale per il riconoscimento della data di nascita in tutte le sue peculiarità;
* realizzazione dell'applicazione.

Settima Settimana

L'obiettivo prefissato è:

* **OD-01**.

Le attività previste sono:

* analisi del meccanismo che permette lo scambio di informazioni con memorizzazione sulla base di quanto già implementato negli assistenti virtuali studiati;
* implementazione del meccanismo analizzato.

Ottava Settimana

In questo ultimo periodo è prevista l'implementazione di eventuali test e la stesura della documentazione finale sull'applicazione realizzata.

2.4 Tecnologie e strumenti utilizzati

2.4.1 Tecnologie

Kotlin

Kotlin è un linguaggio di programmazione fortemente tipizzato, multi-paradigma, multi-piattaforma e open source, sviluppato da Jetbrains nel 2011. È pienamente compatibile con ogni applicazione basata sulla *JVM*[g]. Nel mio caso ho utilizzato Kotlin per costruire un'applicazione Android sotto forma di *PoC* che implementi una particolare funzionalità di Assistant.

ECMAScript

ECMAScript versione 6 abbreviato in *(ES6)* è lo standard di Javascript, un linguaggio di programmazione debolmente tipizzato, orientato agli oggetti e agli eventi che viene utilizzato prevalentemente nella programmazione Web. Nel mio caso ho utilizzato *ES6* per costruire un *PoC* che implementi una particolare funzionalità di Alexa nella sua console dedicata. Inoltre ho realizzato la parte principale dell'applicazione che fa uso della tecnologia Zucchetti.

Swift

Swift è un linguaggio di programmazione orientato agli oggetti e utilizzato esclusivamente per costruire applicazioni per i dispositivi Apple. Nel mio caso l'ho utilizzato per costruire un'applicazione che mi permetta di esplorare una funzionalità di Siri.

HTML

(HTML)[g]è un linguaggio di markup principalmente pensato per pagine o applicazioni basate su web. Io ho utilizzato *HTML5*, l'ultima versione, in quanto è più semplice nella sintassi e non ho necessitato di piena compatibilità con tutti i browser o le versioni meno recenti in un *PoC* sperimentale.

CSS

(CSS)[g]è un linguaggio per la realizzazione di fogli di stile da applicare a pagine o applicazioni basate su web. Io ho utilizzato *CSS3*, l'ultima versione, per l'interfaccia grafica dell'applicazione finale.

jQuery

jQuery è una libreria JavaScript ricca di funzionalità. Permette di eseguire operazioni di spostamento e manipolazione dei documenti *HTML*, gestire degli eventi, creare animazioni e utilizzare Ajax con una *API* di facile utilizzo. Nel mio caso l'ho utilizzata per implementare il riconoscimento dei comandi vocali nell'applicazione.

2.4.2 Strumenti

Android Studio

Android Studio è un ambiente di sviluppo integrato per realizzare applicazioni da utilizzare nella piattaforma Android ed è basato sul software IntelliJ IDEA di JetBrains.

Pubblicato a fine 2014, ha sostituito l'utilizzo dei plug-in specifici di Eclipse per lo sviluppo di applicazioni Android diventando il più diffuso. Durante il mio lavoro è stato ausiliario nell'implementazione del *PoC* per Assistant.
Tra gli strumenti di Android Studio è rilevante App Actions Test Tool che permette di simulare l'utilizzo di Assistant nell'applicazione che si sta sviluppando, in un dispositivo fisico.

Alexa developer console

Alexa developer console è un ambiente di sviluppo di Amazon dedicato alla costruzione delle *Skill* per Alexa. Durante il mio lavoro è stato ausiliario all'implementazione del *PoC* per Alexa.

Xcode

Xcode è un ambiente di sviluppo integrato per realizzare applicazione da utilizzare nelle piattaforme Apple. Durante il mio lavoro è stato ausiliario all'implementazione del *PoC* per Siri.

WebStorm

WebStorm è un ambiente di sviluppo integrato per realizzare applicazioni web con il supporto ad esempio di linguaggi quali *HTML*, *CSS*, *ES6* e *(PHP)*[g]. Durante il mio lavoro è stato ausiliario all'implementazione dell'applicativo finale.

2.5 Motivazioni personali

Durante l'iniziativa StageIT, dedicata agli stage in azienda, ho inizialmente cercato una proposta che trattasse argomenti innovativi, possibilmente di ricerca, rientranti in una delle seguenti tematiche:

* intelligenza artificiale;
* blockchain;
* analisi e previsioni di dati.

Dopo aver consultato diverse azienda, la scelta è ricaduta sulla proposta di Zucchetti. La motivazione principale è stata l'impiego di uno dei rami dell'intelligenza artificiale, la comprensione del linguaggio naturale, che rappresenta parte di una sfida intrapresa da decenni: superare il *Test di Turing*[g]. È un argomento non trattato nel mio percorso di studi ma a mio parere molto interessante.
La seconda motivazione risiede nel contenuto specifico del progetto che ho trovato molto stimolante perché mi prevedeva l'apprendimento e la realizzazione di tecnologie innovative ed in gran parte basato sull'attività di ricerca autonoma.
La terza motivazione risiede nell'azienda stessa in quanto è una delle software house più grandi in Italia, dalle innumerevoli possibilità lavorative in altrettanti ambiti volti anche alla ricerca, con cui ho anche avuto un ottimo rapporto di collaborazione per lo svolgimento del progetto didattico del corso di Ingegneria del Software.

Capitolo 3

Gli assistenti virtuali

3.1 Premessa

Un assistente virtuale è un software capace di interpretare il linguaggio naturale e dialogare con gli utenti che ne fanno uso, eseguendo determinati compiti.
Gli assistenti virtuali analizzati sono Assistant, Alexa e Siri e, nonostante siano software di tre aziende diverse, il loro funzionamento è molto simile: lo sviluppatore deve costruire un'abilità secondo principi analoghi tra loro, addestrare l'intelligenza che la costituisce e permettere all'utente di richiamarla. Queste abilità sono chiamate *Action* per Assistant, *Skill* per Alexa e *Shortcuts* per Siri e sono caratterizzate da nome, frasi di richiamo e azioni da eseguire.
La prima proprietà in comune è il meccanismo di funzionamento delle abilità: gli intenti. Essi consistono in un'azione che soddisfa una richiesta vocale di un utente e nel servizio che la esegue, detto *fulfillment*.
La seconda proprietà in comune è la possibilità di integrare all'interfaccia grafica quella vocale, ove possibile. Questo non viene trattato direttamente nell'analisi ma permetterà di giungere a conclusioni importanti nel seguito.
L'obiettivo dello studio riportato è fornire principi e metodi ad alto livello relativi a progettazione ed implementazione delle sole funzionalità di interesse per l'azienda; tuttavia alcuni dettagli sono stati tralasciati perché ampiamente spiegati nelle pagine di documentazione delle singole tecnologie.

3.2 Assistant

3.2.1 Introduzione

Assistant è l'assistente virtuale di Google ed è capace di riconoscere un comando vocale, elaborarlo attraverso un ragionamento e fornire una risposta. È una tecnologia in continuo miglioramento grazie anche all'immensa mole di dati che Google ha a disposizione per il suo addestramento.

Figura 3.1: Logo di Assistant

Reso pubblico nel 2016, Assistant è ora integrato in tutti i dispositivi con sistema operativo Android, a partire dalla versione 6.0 se hanno a disposizione almeno 1.5 GB di memoria *(RAM)*[g]oppure dalla versione 5.0 se hanno a disposizione almeno 1 GB di memoria *RAM*. È inoltre installabile nei dispositivi con sistema operativo iOS a partire dalla versione 10 e iPadOS; tuttavia l'integrazione è pessima in quanto per richiamarlo è necessario prima invocare Siri. Infine è fruibile nei dispositivi della linea Home e Nest di Google, costruiti e pensati appositamente per ottimizzarne le funzionalità.

3.2.2 Casi d'uso

I casi d'uso trattati sono i seguenti:

* dialogo con l'utente: *Conversational Actions*;
* integrazione di contenuti multimediali pensati per Assistant all'interno di pagine web: *Content Actions*;
* integrazione di Assistant nelle applicazioni Android per eseguire determinate funzionalità: *App Actions*.

3.2.3 Conversational Actions

Descrizione

Le *Conversational Actions* estendono le funzionalità di Assistant con l'obiettivo di creare esperienze di conversazione personalizzate con gli utenti. Esse infatti permettono di gestire le richieste rivolte all'Assistente e costruire le risposte a seguito dell'elaborazione. Una caratteristica importante è la possibilità di comunicare con servizi Web o applicazioni esterne che forniscono una logica di conversazione aggiuntiva grazie agli appositi *SDK*.

Funzionamento

Il principio di funzionamento è illustrato nella seguente figura.

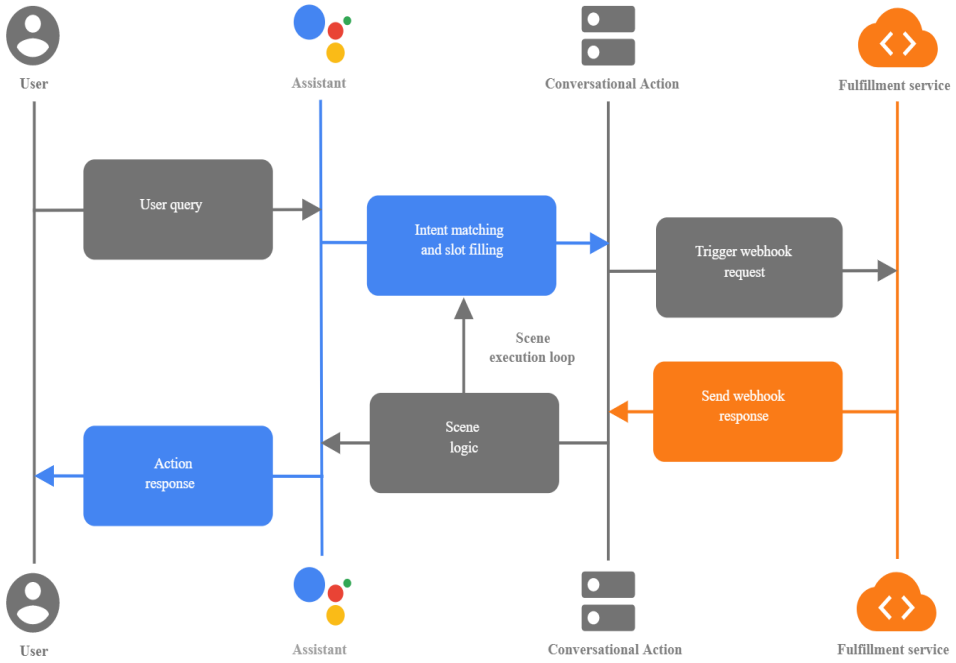

Figura 3.2: Schema di funzionamento delle conversazioni con Assistant

Esso si articola nei seguenti passi:

1. un utente lancia una richiesta sotto forma di comando vocale al dispositivo che ospita Assistant;

2. Assistant che riconosce il comando vocale trasformandolo in stringhe di testo;

3. Assistant invia la stringa riconosciuta ad un server remoto di Google, tramite protocollo *HTTPS*[g], dove risiede la *NLU* per l'elaborazione;

4. la *NLU* del server remoto verifica la possibile corrispondenza tra la stringa ricevuta e l'insieme di frasi che lo sviluppatore ha inserito durante la costruzione della propria abilità;

5. se non vi è alcuna corrispondenza viene subito ritornata una risposta negativa e riferito all'utente. Si ricomincia quindi dal punto 1;

6. se una corrispondenza ha dato esito positivo viene selezionato l'intento e attivato il suo servizio di *fulfillment* che si occupa dell'esecuzione;

7. la conclusione prevede il ritorno a cascata delle chiamate effettuate ed infine la costruzione e l'invio della risposta al dispositivo mittente;

8. il dispositivo riferisce la riposta all'utente che potrà poi procedere con una nuova richiesta fino al termine dell'esecuzione dell'abilità oppure interrompere forzatamente la conversazione.

Qualora l'utente effettuasse una richiesta di invocazione, prima di scegliere l'intento da eseguire viene cercata una corrispondenza tra le *Action* a disposizione per capire quale avviare.
Infine, affinché la *Action* tenga un comportamento adeguato, è necessario applicare correttamente i principi di costruzione e svolgere una consistente attività di addestramento durante lo sviluppo.

Progettazione

Nella costruzione di una *Action* la prima attività da svolgere è l'analisi dei requisiti ovvero comprendere dettagliatamente il comportamento che si vuole ottenere. Essa però non viene trattata in quanto è legata ai processi interni aziendali.
L'attività successiva, invece, è la progettazione ed è mirata a tre aspetti:

* modalità di invocazione;
* tipologia e formato delle richieste accettate;
* tipologia e formato delle risposte che l'utente si aspetta.

Per la progettazione di richieste e risposte è necessario ragionare sullo scopo dell'*Action* che si vuole implementare e svolgere un'analisi statistica e probabilistica sulle frasi che l'utente potrebbe pronunciare o aspettarsi dall'assistente, cercando di rendere la conversazione più naturale possibile. Durante l'esecuzione della *build*[g]dell'*Action*, Assistant sarà addestrato sulle frasi immesse al fine di interpretarle correttamente.
Per la modalità di invocazione, invece, Google fa una distinzione:

* invocazione esplicita;
* invocazione implicita.

L'invocazione esplicita è la più comunemente utilizzata e consiste nell'esprimere una frase che riporti la seguente struttura:

1. parola di attivazione: "Hey Google" oppure "Ok Google";
2. parola di avvio: chiedi, fai, dimmi, raccontami e vocaboli simili;
3. nome di invocazione: nome deve identifica la *Action*;
4. tips: parametri aggiuntivi, possibilmente opzionali, implementati come variabili che specificano ulteriormente la richiesta dell'utente;
5. elementi aggiuntivi: parole addizionali che l'utente può pronunciare con lo scopo di contestualizzare o precisare il dominio della richiesta.

Grazie a questa struttura fissa, Assistant riesce a comprendere quale *Action* attivare per avviare la conversazione.
L'invocazione implicita, invece, si verifica quando l'utente effettua una richiesta senza aver esplicitato l'*Action* o l'intento da eseguire. In questo caso la business logic ha il compito di comprendere la richiesta e associare l'*Action* che ritiene più corretta; qualora non ne trovasse alcuna, effettuerà una ricerca in Internet inserendo come testo la richiesta stessa e ritornerà i risultati come risposta. Tuttavia il funzionamento di questa modalità non è garantito da Google in quanto è ancora in via di sviluppo e richiede come condizione necessaria ma non sufficiente che lo sviluppatore abbia inserito un numero di frasi ampio e completo per l'addestramento.

Implementazione

L'attività che segue è l'implementazione e per svolgerla Google offre due strumenti:

* Dialogflow;
* Conversational Actions SDK.

Dialogflow è uno strumento utilizzato per creare conversazioni personalizzate in modo semplice ed intuitivo. Si basa sulla *NLU* di Assistant e si appoggia ad un *webhook* per la gestione dei dati, *Firebase*[g] è quello predefinito.
Per costruire una *Conversational Action* necessita di un agente ovvero un modulo di comprensione del linguaggio naturale che gestisce le conversazioni con gli utenti sgravando lo sviluppatore da numerosi oneri; deve quindi essere creato prima di iniziare l'effettiva costruzione. A causa dei limiti imposti dall'account *Firebase* disponibile non si è potuto approfondire il suo funzionamento come invece si è fatto per Alexa con uno strumento simile fornito da Amazon; tuttavia i principi di funzionamento e implementazione sono molto simili.
Le Conversational Actions SDK consistono in un'interfaccia *HTTPS* che permette di costruire ed eseguire le *Conversational Actions* all'interno di una propria applicazione per elaborare le richieste effettuate. Grazie ad essa infatti Assistant può comunicare con applicazioni terze e il requisito che ne deriva è possedere una propria *NLU* per la comprensione del linguaggio naturale.
L'architettura ad alto livello del sistema che rappresenta l'utilizzo delle *Conversational Actions* attraverso *SDK* è così composta:

* dispositivo fisico con Assistant integrato che riceve la richiesta vocale dell'utente;
* *NLU* di Google che ha il compito di riconoscere la richiesta dell'utente e associare l'intento corretto;
* servizio cloud remoto che, ricevuta la richiesta di esecuzione dell'intento, esegue il servizio di *fulfillment* associato e ritorna la risposta alla *NLU* che a sua volta la ritorna al dispositivo.

Le componenti di un agente di Dialogflow oppure una *Conversational Actions* costruita con le *SDK* sono uguali:

* Default Actions: azione cui corrisponde l'evento chiamato *GOOGLE_ASSISTANT_WELCOME* per Dialogflow e *actions.intent.MAIN* per le *SDK* che rappresenta la prima interazione con l'*Action*. Una condizione necessaria che la caratterizza è l'esistenza di uno ed un solo intento per gestire questo evento. La risposta predefinita è statica e preconfigurata ma è comunque possibile renderla dinamica costruendo un servizio di *fulfillment* che la compone a tempo di esecuzione;
* Addictional Actions: azione aggiuntiva alla Default Actions utilizzata per aggiungere e specificare le capacità dell'*Action* stessa. Possono essere molteplici e sono automaticamente indicizzate per l'invocazione implicita.

Ognuna di queste componenti corrisponde ad un intento che l'*Actions* può gestire.
Una volta stabilite queste componenti, è necessario definire l'interfaccia della propria conversazione e per poterlo fare si deve creare un intento definendo:

* nome;

* contesto;
* evento scatenante;
* frasi di input per l'addestramento;
* azioni da eseguire;
* eventuali parametri aggiuntivi per la conversazione e formato della risposta.

Le azioni da eseguire corrispondono alla creazione di un *fulfillment* che fornisce la logica per processare la richiesta dell'utente e generare una risposta.
Qui emerge una grande differenza tra Dialogflow e *SDK*: mentre il primo utilizza la *NLU* di Google rivelandosi quindi di poco interesse per i progetti dell'azienda, il secondo prevede l'utilizzo obbligatorio di una *NLU* proprietaria dello sviluppatore che invece si è rivelato utile per l'azienda.

Comunicazione

Lo scambio di dati tra il dispositivo che interagisce direttamente con l'utente e la *NLU* di Assistant avviene tramite oggetti JSON di cui però non viene fornita la struttura nella documentazione.
Lo scambio di dati che invece avviene tra la *NLU* di Assistant e quella del proprio applicativo, diretta conseguenza dell'utilizzo delle *SDK*, prevede un metodo di comunicazione dedicato. Google perciò impone l'utilizzo di oggetti JSON con una struttura fissata in cui campi dati più importanti sono:

* isInSandbox: attributo booleano, se vale true indica l'utilizzo in un ambiente di test;
* Surface: oggetto che contiene la modalità di interazione, scelta tra solo testuale, solo visiva e multimediale;
* Inputs: oggetto che contiene l'insieme degli input specificati per la *Actions* tra cui la richiesta dell'utente sotto forma di stringa testuale;
* User: oggetto che contiene le informazioni dell'utente che ha eseguito la richiesta;
* Device: oggetto che contiene le informazioni del dispositivo da cui è arrivata la richiesta;
* Conversation: oggetto che contiene i dati strettamente legati alla conversazione in corso tra cui conversationId che la identifica univocamente e conversationToken che salva i dati durante la conversazione.

Di particolare rilevanza è il salvataggio dei dati durante la conversazione in quanto permette di avere sempre a disposizione determinati dati, possibilmente importanti, per la corretta esecuzione dell'*Action* ma soprattutto di dare all'utente la sensazione di interagire con un sistema intelligente che ha capacità di memoria. Quest'ultima caratteristica è molto importante perché consente di definire e mantenere il contesto della conversazione, qualunque esso sia, incrementando notevolmente la qualità dell'esperienza d'uso.
Per capire quali elementi devono essere salvati, Assistant utilizza delle variabili all'interno delle frasi di richiesta dette *tips*. Esse sono definite dallo sviluppatore durante la progettazione e, quando l'utente pronuncia parole o dati in una posizione all'interno

della frase corrispondente a quella di una variabile, vengono automaticamente salvati nel campo chiamato *conversationToken*. Il loro limite è rappresentato dalla conversazione stessa: al termine tutti i dati salvati vengono persi. Perciò, se si vuole una persistenza duratura nel tempo, bisogna utilizzare una struttura esterna permanente come ad esempio un database.

3.2.4 Content Actions

Descrizione

Le *Content Actions* sono delle funzionalità che estendono Assistant ma integrate nelle pagine Web. In particolare, inserendo dati strutturati al loro interno, permettono di costruire *Actions* che ne presentano il contenuto in modo interattivo a seguito di un comando vocale.

Funzionamento

Il loro funzionamento verte esclusivamente sull'algoritmo di Google ed piuttosto semplice: l'utente esegue una ricerca in internet con un comando vocale lanciato ad Assistant e, se l'algoritmo di Google trova una *Content Actions* che ha corrispondenza con la richiesta effettuata, il risultato sarà la risposta della *Action* stessa.

Dati strutturati

Le *Content Actions* sono basate su dati strutturati. Essi corrispondono ad un oggetto JSON iniettato dinamicamente in una pagina *HTML* e contengono un insieme di metadati che definiscono le proprietà del contenuto multimediale che si vuole creare. Alcune di esse sono obbligatorie perché vengono utilizzate da Assistant per accedere al contenuto, molte altre sono opzionali ma consigliate per ottenere una migliore indicizzazione nel motore di ricerca.
Esistono inoltre due formati alternativi agli oggetti JSON ma poco raccomandati:

* microdata: tag HTML che permette di includere dati strutturati al suo interno;
* RDFa: estensione di HTML5 che permette di definire contenuti per i motori di ricerca e in particolare per l'indicizzazione.

Sono messi a disposizione principalmente per compatibilità con sistemi non tradizionali.

Costruzione

Per costruire una *Content Actions* è necessaria una progettazione mirata della pagina e a tal proposito Google offre un insieme di template pensati per le attività che più si prestano: podcasts ricette, news, *How-to guides* e FAQs.
Durante la realizzazione della pagina, invece, si devono inserire i dati strutturati in base alle proprie esigenze ed effettuare dei test per verificarne la corretta indicizzazione attraverso gli strumenti forniti.
È infine importante notare che Google non garantisce il richiamo della propria *Actions* in nessuna condizione: l'algoritmo di ricerca, infatti, ha come obiettivo principale fornire all'utente i migliori risultati possibili nel momento in cui viene eseguita la ricerca e ciò può non corrispondere con la propria *Actions*.

3.2.5 App Actions

Descrizione

Le *App Actions* sono delle funzionalità aggiuntive di Assistant che interagiscono esclusivamente con applicazioni Android ma ancora in via di sviluppo. In particolare permettono agli utenti di eseguire il *deep linking* delle funzionalità specifiche di un'applicazione attraverso un comando vocale ed il loro risultato può manifestarsi sotto forma di risposta vocale oppure grafica, facendo opzionalmente uso delle *Android Slides*. Infine è importante notare che le *App Actions* sono per ora fornite in anteprima e supportano solo la lingua inglese americana.

Funzionamento

Il funzionamento delle *App Actions* è rappresentato dalla seguente figura.

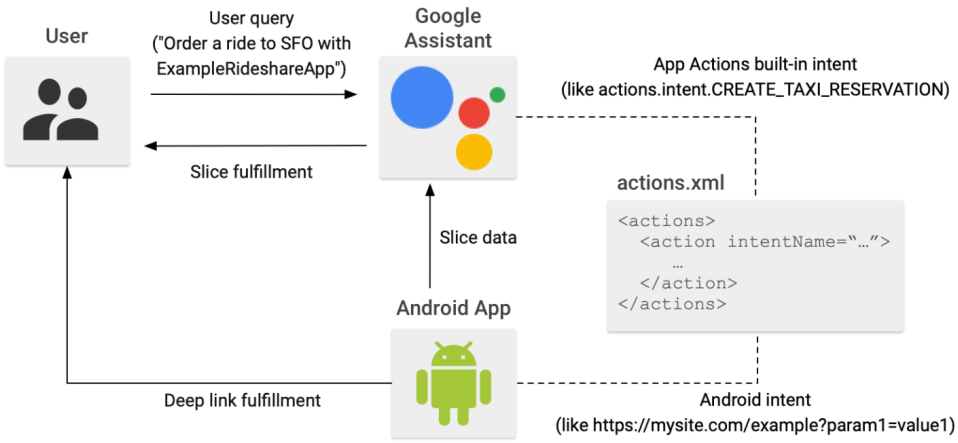

Figura 3.3: Schema di funzionamento delle App Actions

Viene descritto dai seguenti passi:

1. un utente invoca una *App Actions* attraverso un apposito comando vocale;
2. Assistant riconosce il comando vocale trasformandolo in stringhe di testo;
3. Assistant invia la stringa riconosciuta ad un server remoto di Google, tramite protocollo *HTTPS*, dove risiede la *NLU* per l'elaborazione;
4. la *NLU* del server remoto verifica la possibile corrispondenza tra la stringa ricevuta e la frase che lo sviluppatore ha predisposto per l'attivazione;
5. se non vi è alcuna corrispondenza l'assistente eseguirà una ricerca in Internet della richiesta ritornandone l'esito come risposta;
6. se la corrispondenza ha dato esito positivo viene selezionato l'intento associato con il suo servizio di *fulfillment* che si occupa dell'esecuzione;
7. la conclusione prevede la risposta all'utente in termini di funzionalità eseguita.

Progettazione

La progettazione di una *App Actions* presuppone l'esistenza di un'applicazione costruita per Android a cui aggiungerla. Gli elementi cardine sono la decisione della frase di invocazione in quanto rappresenta l'unica modalità di attivazione della *Action* e la scelta dell'intento stesso in quanto deve essere il più adeguato a ciò che si vuole realizzare. Google mette a disposizione degli intenti preconfigurati e i macro argomenti che trattano sono:

* attivazione funzionalità delle applicazioni;
* esecuzione degli ordini online;
* operazioni in ambito finanziario;
* ordinazione di cibi e bevande da un menù preconfigurato;
* monitoraggio di salute e fitness;
* richiesta di trasporto tramite Taxi.

Ad oggi non è ancora possibile costruire un intento personalizzato.

Implementazione

Dopo aver progettato la *Action* si passa all'implementazione. Questa attività consiste nell'associare l'intento ed il suo *fulfillment* all'applicazione. Più in dettaglio si deve:

* registrare il proprio intento nel file *actions.xml* mappando gli eventuali parametri con il servizio di *fulfillment* collegato;
* aggiornare il file *AndroidManifest.xml* inserendo le dipendenze con gli intenti designati.

Dopo aver svolto questi compiti è necessario associare la *Action* con il plug-in *App Actions Test Tool* integrato in Android Studio ed eseguire almeno un test; in caso contrario non è possibile richiamarla.

3.2.6 Proof of concept

Analisi dei requisiti

Per il proof of concept relativo ad Assistant, in comune accordo con il tutor e sulla base delle ricerche effettuate, è stato scelto di implementare una *App Action* sugli intenti legati al fitness. Il suo scopo è verificare la fattibilità e le potenzialità delle *App Actions* con un esempio concreto nonostante il loro attuale supporto alla sola lingua inglese. Ho quindi deciso di realizzare un timer con un'interfaccia grafica minimale che calcoli il tempo impiegato nello svolgimento di attività fisica per ogni sport disponibile negli intetni preconfigurati, con la possibilità di attivarlo tramite comando vocale.

Implementazione

Nell'attività di implementazione ho seguito le linee guida fornite da Google. Inizialmente ho realizzato un'applicazione Android che presentasse tutte le funzionalità previste, fruibili tramite l'interfaccia grafica permettendo all'utente di attivare un

timer dall'apposito pulsante, fermare l'attività in qualunque momento e visualizzare i risultati nello storico. A questo punto ho registrato gli intenti scelti nel file *actions.xml* e aggiunte le dipendenze nel file *AndroidManifest.xml* ed infine ho sviluppato il codice che deve essere eseguito al lancio del comando vocale.

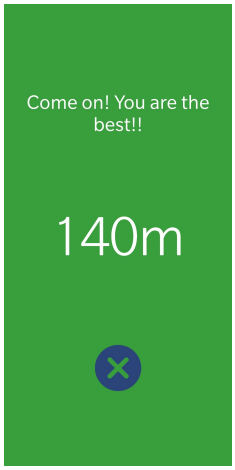

Figura 3.4: Esempio di funzionamento del PoC con Assistant

Test

Per quanto riguarda i test ho configurato il plug-in *App Actions Test Tool* che permette di eseguire le verifiche su un dispositivo fisico, in quanto il supporto al simulatore non è ancora attivo, e di associare la mia *Action* ad Assistant.

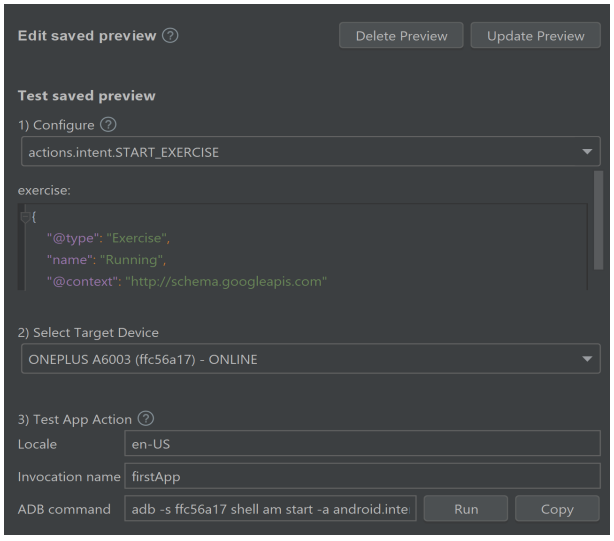

Figura 3.5: App Actions Test Tool

3.3 Alexa

3.3.1 Introduzione

Alexa è l'assistente virtuale di Amazon ed è capace di riconoscere un comando vocale, elaborarlo attraverso un ragionamento e fornire una risposta. È una tecnologia in continuo sviluppo grazie anche alla consistente mole di dati che Amazon ha a disposizione per il suo addestramento.

Figura 3.6: Logo di Alexa

La prima versione di Alexa risale al 2014 e da allora ha fatto notevoli miglioramenti. È integrato in tutti i dispositivi della linea Amazon Echo costruiti appositamente per ottimizzarne l'utilizzo; tuttavia è installabile in tutti i dispositivi con sistema operativo Android in versione 5.0 o maggiore, iOS in versione 9.0 o maggiore e iPadOS.

3.3.2 Casi d'uso

Il caso d'uso trattato è: dialogo con l'utente attraverso le *Skill*.

3.3.3 Skill di conversazione

Descrizione

Le *Skill* consistono in funzionalità personalizzate e aggiuntive per Alexa mirate a migliorare l'esperienza d'uso degli utenti. Attraverso le *Skill* lo sviluppatore può ricevere le richieste rivolte ad Alexa, soddisfarle e restituire una risposta. Una caratteristica importante è la possibilità di comunicare con servizi Web o applicazioni esterne che forniscono una logica di conversazione aggiuntiva grazie alle *API* presenti nell'Alexa Skill Kit.
L'obiettivo principale delle *Skill* è permettere all'utente una conversazione finalizzata a soddisfare un suo bisogno; tuttavia non è possibile eseguire applicazioni o anche solo funzionalità al loro interno.

Funzionamento

Il principio di funzionamento si articola nei seguenti passi:

1. un utente lancia un comando vocale al dispositivo che ospita l'assistente;
2. Alexa riconosce le parole pronunciate trasformandole in stringhe di testo;
3. Alexa invia la stringa riconosciuta ad un server remoto per l'elaborazione;

4. il server remoto attiva la propria *NLU* che verifica la possibile corrispondenza tra la stringa ricevuta e l'insieme di frasi che lo sviluppatore ha inserito nella propria abilità;
5. se la ricerca delle corrispondenze ha dato esito negativo viene riferita all'utente la mancata comprensione oppure viene data la risposta di una ricerca su Internet della richiesta stessa;
6. se la ricerca delle corrispondenze ha dato esito positivo viene selezionato l'intento;
7. prima di eseguire il codice dell'intento vengono invocati gli eventuali *Request Interceptors* definiti dallo sviluppatore;
8. viene richiamato gestore dell'intento, rappresentante il codice da eseguire, che porterà a termine l'intento;
9. dopo aver gestito l'evento, vengono richiamati gli eventuali *Response Interceptors* definiti dallo sviluppatore;
10. viene costruita la risposta e ritornata al dispositivo che ospita l'assistente;
11. il dispositivo riferisce la riposta all'utente che potrà poi procedere con una nuova richiesta, fino al termine dell'esecuzione dell'abilità previsto dal programmatore durante la costruzione, oppure interrompere forzatamente la conversazione.

Progettazione

Nella costruzione di una *Skill* la prima attività da svolgere è l'analisi dei requisiti ovvero comprendere dettagliatamente il comportamento che si vuole ottenere. Essa però non viene trattata in quanto è legata ai processi interni aziendali.
L'attività successiva, invece, è la progettazione che deve essere mirata a tre aspetti:

* tipologia e formato delle richieste accettate;
* tipologia e formato delle risposte che l'utente si aspetta;
* modalità di invocazione.

Per la progettazione di richieste e risposte è necessario ragionare sullo scopo della *Skill* che si vuole implementare e svolgere un'analisi statistica e probabilistica sulle frasi che l'utente potrebbe pronunciare o aspettarsi dall'assistente. L'obiettivo infatti è rendere la conversazione più naturale possibile.
Per la modalità di invocazione, anche Amazon fa una distinzione:

* invocazione esplicita;
* invocazione implicita.

L'invocazione esplicita è la più comunemente utilizzata e consiste nell'esprimere una frase che riporti la seguente struttura:

1. parola di attivazione: "Alexa";
2. parola di avvio: chiedi, fai, dimmi, raccontami e vocaboli simili;
3. nome di invocazione: nome deve identifica la *Skill*;

4. slots: parametri aggiuntivi, possibilmente opzionali, implementati come variabili che specificano ulteriormente la richiesta dell'utente;

5. elementi aggiuntivi: parole addizionali pronunciate dall'utente con lo scopo di contestualizzare o precisare il dominio della richiesta.

Grazie a questa struttura fissa, Alexa è in grado di comprendere quale *Skill* attivare per avviare la conversazione.
L'invocazione implicita, invece, si verifica quando l'utente effettua una richiesta senza aver esplicitato la *Skill* o l'intento da eseguire. In questo caso la business logic di Alexa deve comprendere la richiesta e associare la *Skill* che ritiene più corretta; qualora non ne trovasse alcuna, effettuerà una ricerca in Internet inserendo come testo la richiesta stessa e ritornerà come risposta i risultati. Tuttavia il funzionamento di questa modalità non è garantito da Amazon in quanto è ancora in uno stato embrionale e richiede, come condizione necessaria ma non sufficiente, che lo sviluppatore abbia inserito un numero di frasi ampio e completo per l'addestramento.

Implementazione

L'attività che segue è l'implementazione ed in merito a ciò Amazon offre due strumenti:

* Alexa Developer Console;

* Alexa SDK.

Il primo è Alexa Developer Console, uno strumento che integra Alexa Skills Kit e fornisce un'interfaccia allo sviluppare per creare *Skill* personalizzate di diverse tipologie, tra cui quelle di conversazione, in modo semplice e intuitivo. Si basa sulla *NLU* di Alexa e si appoggia ad AWS Lambda come *webhook* predefinito per la gestione dei dati.
Per implementare una *Skill* necessita l'esecuzione di due macro compiti:

* costruire un modello di interazione;

* implementare la logica interna.

Costruire un modello di interazione significa definire l'interfaccia vocale e gli intenti che si vogliono implementare. Più in dettaglio si intende inserire nell'interfaccia vocale le possibili frasi di invocazione, richiesta e risposta oltre ad eventuali slot e formalizzare le azioni che la *Skill* deve essere capace di eseguire sotto forma di intento.
È possibile inoltre scegliere tra gli intenti preconfigurati da Amazon e quelli personalizzati. Gli ultimi sono interamente a carico dello sviluppatore e necessitano la definizione di:

* nome;

* contesto;

* evento scatenante;

* frasi di input per l'addestramento;

* azioni da eseguire;

* eventuali parametri aggiuntivi per la conversazione e formato della risposta.

Non ci sono vincoli sull'utilizzo di uno piuttosto che dell'altro e infatti possono anche essere usati contemporaneamente.
Implementare la logica interna, invece, significa implementare il codice che definisce il comportamento dei singoli intenti e quindi della *Skill* nel suo complesso.
Il secondo strumento disponibile è *Alexa SDK* e consiste a sua volta in un insieme di strumenti che permettono allo sviluppatore di interagire con Alexa da un'applicativo esterno che possiede una propria *NLU*. Essi sono disponibili nei linguaggi Javascript/-Typescript, Java e Phyton. I principi di progettazione e implementazione sono gli stessi di quelli analizzati per Alexa Developer Console con la sola differenza che, utilizzando Alexa SDK, è necessario aggiungere uno strato di comunicazione tramite oggetti JSON per lo scambio di dati tra Alexa ed il proprio applicativo.

Interceptors

Per implementare le *Skill*, esclusivamente per Alexa, sono offerte funzioni speciali di richiesta e risposta che vengono elaborate immediatamente prima e dopo la gestione degli intenti e sono dette *Interceptors*. È possibile utilizzarli per richiamare la logica comune che si applica a più richieste o risposte con l'obiettivo di evitare la duplicazione del codice.
I *Request Interceptors* sono richiamati subito prima dell'esecuzione di un intento e permettono di aggiungere qualsiasi logica che deve essere eseguita per ogni richiesta, indipendentemente dalla tipologia.
I *Response Interceptors* sono richiamati subito dopo l'esecuzione di un intento. Anch'essi permettono di aggiungere qualsiasi logica che deve essere eseguita per ogni risposta ma solitamente sono utilizzate per la sanificazione della risposta, l'internazionalizzazione e la validazione.

Comunicazione

Lo scambio di dati tra il dispositivo che interagisce direttamente con l'utente e la *NLU* di Alexa avviene tramite oggetti JSON di cui però non viene fornita la struttura nella documentazione.
Lo scambio di dati che invece avviene tra la *NLU* di Alexa e quella del proprio applicativo, diretta conseguenza dell'utilizzo delle *SDK*, prevede un metodo di comunicazione dedicato. Amazon ha deciso di utilizzare gli oggetti JSON come mezzo per lo scambio di dati con strutture fissate ma differenti tra richiesta e risposta.
I campi principali dell'oggetto di richiesta sono:

* session: oggetto che fornisce informazioni riguardanti il contesto associato alla richiesta. È disponibile solo per conversazioni che non contengono contenuti multimediali;
* context: oggetto che fornisce le informazioni riguarrdanti lo stato della conversazione corso, del servizio di Alexa in esecuzione e del dispositivo con cui interagisce l'utente;
* request: oggetto che fornisce i dettagli della richiesta utente.

I campi principali dell'oggetto di risposta sono:

* sessionAttributes: mappa chiave-valore di tutti i dati di sessione;

* response: oggetto che definisce che cosa il dispositivo deve renderizzare per rispondere all'utente.

Di particolare rilevanza è il salvataggio dei dati durante la conversazione che permette di richiedere determinati dati possibilmente importanti per la corretta esecuzione della *Skill* ma soprattutto di dare all'utente la sensazione di interagire con un'intelligenza che ha capacità di memoria. Quest'ultima caratteristica è molto importante perché consente di definire e mantenere il contesto della conversazione, qualunque esso sia, incrementando notevolmente la qualità dell'esperienza d'uso dell'utente.
Per capire quali elementi devono essere salvati, Alexa utilizza delle variabili all'interno delle frasi di richiesta dette *slots*. Esse sono definite dallo sviluppatore durante la progettazione così che, quando l'utente pronuncia parole o dati in una posizione all'interno della frase corrispondente a quella di una variabile, vengono automaticamente salvati nel campo chiamato conversationToken. Il loro limite è rappresentato dalla conversazione stessa: al suo termine tutti i dati salvati vengono persi. Perciò, se si vuole una persistenza duratura nel tempo, bisogna utilizzare una struttura di supporto esterna come ad esempio un database.

3.3.4 Proof of concept

Analisi dei requisiti

Per il *PoC* relativo ad Alexa, in comune accordo con il tutor e sulla base delle ricerche effettuate, è stato scelto di implementare una *Skill* in grado di riconoscere la data di nascita di una persona. Il suo scopo è verificare le capacità conversazionali di Alexa e comprendere il meccanismo di funzionamento da loro adottato con un esempio concreto. Ho quindi deciso di realizzare un'interfaccia vocale in grado di comprendere un insieme di frasi preconfigurate per esprimere la propria data di nascita e un insieme di frasi per porre delle domande qualora l'utente fornisca una risposta incompleta o errata.

Implementazione

Per implementare la *Skill* ho deciso di utilizzare la console fornita da Amazon che maschera la complessità delle *SDK* e della comunicazione con esse. Il primo step è stato definire il modello di interazione composto da:

* frasi per la comunicazione con l'utente;

* intento per associare l'azione da eseguire;

* slot per le variabili.

Inizialmente è stato definito l'insieme delle frasi possibili per le richieste dell'utente e le risposte di Alexa. Successivamente è stato implementato un intento personalizzato identificato dal nome *RegisterBirthdayIntent* in cui sono stati definiti gli slot della conversazione: giorno, mese e anno.

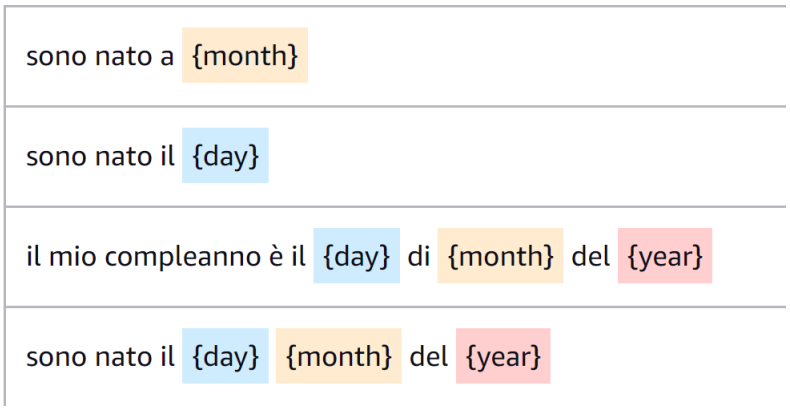

Figura 3.7: Esempio frasi RegisterBirthdayIntent

Infine è stato abilitato il meccanismo di richiesta degli slot in caso di mancanza. In questo modo, sulla base delle frasi che ho configurato, Alexa può continuare a chiedere all'utente giorno, mese e anno finché non saranno forniti.


```
{
    "body": {
        "version": "1.0",
            "response": {
            "outputSpeech": {
                "type": "SSML",
                    "ssml": "<speak>Il tuo compleanno è il 9 di agosto, 1998.</speak>"
            },
            "type": "_DEFAULT_RESPONSE"
            },
        "sessionAttributes": {},
        "userAgent": "ask-node/2.8.0 Node/v10.20.0 sample/happy-birthday/mod3"
    }
}
```


Figura 3.8: Esempio JSON di risposta PoC Alexa

Test

Per eseguire i test è disponibile uno strumento all'interno della console che integra Alexa; tuttavia non ne permette l'automatizzazione.

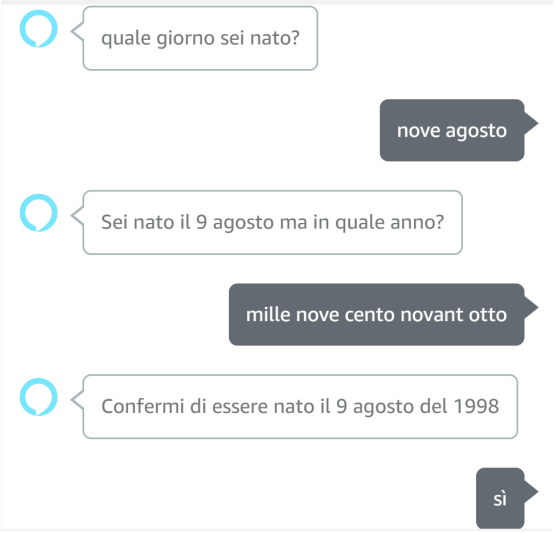

Figura 3.9: Esempio funzionamento PoC Alexa

3.4 Siri

3.4.1 Introduzione

Siri è l'assistente virtuale di Apple ed è capace di riconoscere un comando vocale, elaborarlo attraverso un ragionamento e fornire una risposta. È una tecnologia in continuo sviluppo grazie anche alla contingente mole di dati a disposizione di Apple per il suo addestramento.

Figura 3.10: Logo di Siri

La prima versione è stata introdotta in iOS 5 nel 2012, senza ancora offrire il supporto a tutte le lingue e a tutti i dispositivi. Ora invece è integrato in Homepod e tutti i dispositivi con sistema operativo iOS versione 8.0 o superiore, iPadOS, watchOS, tvOS e MacOS versione 10.12 o superiore. Rimane comunque un esclusiva di Apple e non è installabile in altri sistemi.

3.4.2 Casi d'uso

Il caso d'uso trattato è: esecuzione di operazioni specifiche nelle applicazioni attraverso le *Shortcuts*.

3.4.3 Shortcuts

Descrizione

Le *Shortcuts* consistono in funzionalità aggiuntive per Siri e migliorano l'esperienza d'uso dei dispositivi da parte degli utenti. In particolare è presente un'applicazione si sistema chiamata shortcuts (*comandi* se le impostazioni sono in lingua italiana) al cui interno, per ogni altra applicazione installata nel proprio dispositivo, sono visualizzate tutte le frasi messe a disposizione dagli sviluppatori per eseguire operazioni tramite Siri. L'utente può inoltre personalizzare tali frasi per renderle più intuitive e facili da ricordare.
L'obiettivo principale delle *Shortcuts* è migliorare e personalizzare l'interattività con le applicazioni dell'ecosistema Apple mentre l'aspetto conversazionale risulta finalizzato solo allo scopo principale.

Funzionamento

Il principio di funzionamento si articola nei seguenti passi:

1. un utente lancia un comando vocale al dispositivo che ospita Siri;
2. Siri riconosce le parole pronunciate trasformandole in stringhe di testo;
3. Siri invia la stringa riconosciuta ad un server remoto per l'elaborazione;
4. La *NLU* presente nel server remoto verifica una possibile corrispondenza tra la stringa ricevuta e l'insieme di frasi che lo sviluppatore ha inserito come *Shortcuts* per la propria applicazione;
5. se la ricerca delle corrispondenze ha dato esito negativo viene riferita all'utente la mancata comprensione oppure viene data la risposta di una ricerca su Internet della richiesta stessa;
6. se la ricerca della corrispondenza ha dato esito positivo viene selezionato l'intento;
7. viene richiamata la porzione di codice che porterà a termine l'intento;
8. viene costruita e ritornata la risposta al dispositivo che ospita l'assistente;
9. il dispositivo riferisce la riposta all'utente che potrà procedere con una nuova richiesta fino al termine dell'esecuzione dell'abilità.

Progettazione

Nella costruzione di una *Shortcuts* la prima attività da svolgere è l'analisi dei requisiti ovvero comprendere dettagliatamente il comportamento che si vuole ottenere. Essa però non viene trattata in quanto è legata ai processi interni aziendali.
L'attività successiva, invece, è la progettazione che deve essere mirata a tre aspetti:

* tipologia e formato delle richieste accettate;
* tipologia e formato delle risposte che l'utente si aspetta;
* modalità di invocazione.

Per richieste e risposte è necessario ragionare sullo scopo della *Shortcuts* che si vuole implementare e svolgere un'analisi statistica e probabilistica sulle frasi che l'utente potrebbe pronunciare o aspettarsi dall'assistente. In questo caso le frasi di richiesta assumono un'importanza minore in quanto una delle funzionalità su cui Apple punta molto è fornire all'utente la possibilità di personalizzarle dall'applicazione *Shortcuts*. L'obiettivo primario perciò si sposta dalla conversazione alla capacità di soddisfare un la richiesta dell'utente mentre assume un ruolo meno rilevante rendere la conversazione naturale. Durante l'esecuzione della *build* dell'applicazione, Siri sarà automaticamente addestrato sulle frasi immesse per interpretarle correttamente.
Per la modalità di invocazione, anche Apple fa una distinzione:

* invocazione esplicita;
* invocazione implicita.

L'invocazione esplicita è la più comunemente utilizzata e consiste nell'esprimere una frase che riporti la seguente struttura:

1. parola di attivazione: "Hey Siri";
2. parola di avvio: chiedi, fai, dimmi, raccontami e vocaboli simili;
3. nome di invocazione: nome deve identifica la *Shortcuts*;
4. parametri: parametri aggiuntivi, possibilmente opzionali, implementati come variabili che specificano ulteriormente la richiesta dell'utente;
5. elementi aggiuntivi: parole addizionali pronunciate dall'utente con lo scopo di contestualizzare o precisare il dominio della richiesta.

Grazie a questa struttura fissa, Siri è in grado di comprendere quale *Shortcuts* attivare per avviare la conversazione.
L'invocazione implicita, invece, si verifica quando l'utente effettua una richiesta senza aver esplicitato l'intento da eseguire. In questo caso la business logic deve comprendere la richiesta e associare la *Shortcuts* che ritiene più corretta; qualora non ne trovasse alcuna, effettuerà una ricerca in Internet inserendo come testo la richiesta stessa e ritornerà come risposta i risultati. Tuttavia il funzionamento di questa modalità non è garantito da Apple in quanto è ancora in uno stato embrionale e richiede, come condizione necessaria ma non sufficiente, che lo sviluppatore abbia inserito un numero di frasi ampio e completo per l'addestramento.

Implementazione

L'attività che segue è l'implementazione ed in merito a ciò Apple offre uno strumento: Sirikit. Più in dettaglio è un insieme di strumenti che forniscono un'interfaccia per costruire interazioni tra Siri e le applicazioni. Le *Shortcuts* si basano sulla *NLU* di Siri e prevedono l'utilizzo obbligatorio di Xcode in quanto devono essere inserite nel progetto designato per la loro integrazione.
Apple è da sempre molto pignola in materia di autorizzazioni, sia per l'utilizzo dei propri strumenti sia per il software. Quindi, per implementare una *Shortcuts*, è prevista prima una serie di passaggi:

* abilitare la Capability di Siri nel proprio progetto di Xcode;

* configurare il file Info.plist includendo una chiave il cui valore è una stringa che descrive quali informazioni l'applicazione condivide con SiriKit;

* richiedere l'autorizzazione dell'applicazione iOS. Per farlo è necessario includere il metodo *requestSiriAuthorization(_:)* della classe INPreferences immediatamente dopo il codice che avvia l'applicazione. Grazie a ciò appare il prompt che fa scegliere all'utente se autorizzare o negare l'applicazione all'utilizzo di Siri. È comunque possibile cambiare tale scelta nelle impostazioni del dispositivo.

La maggior parte delle interazioni possibili tramite SiriKit è gestita dalle *App Extension* ovvero estensioni delle funzionalità predefinite per un'applicazione sotto forma di intento. Queste si suddividono in due tipologie:

* *Intent App Extension*: l'utente effettua una richiesta, essa viene ricevuta dall'applicazione che successivamente seleziona l'intento corretto per soddisfarla;

* *Intent UI App Extension*: consiste in un *intent App Extension* come la precedente in cui però, dopo aver soddisfatto la richiesta dell'utente, visualizza i contenuti in una finestra personalizzata. È un arricchimento non obbligatorio che si pone l'obiettivo di migliorare ll'esperienza d'uso dell'utente.

Il principio di costruzione è uguale per entrambe con l'ovvia differenza che per la seconda è necessario provvedere ad un'interfaccia grafica aggiuntiva. I passi sono quindi i seguenti:

* verificare che il procedimento di autorizzazione sia stato eseguito correttamente. Questo è possibile farlo controllando tramite Xcode che la Capability di Siri sia abilitata;

* aggiungere un *Intents App Extension* (o *Intent UI App Extension*) al progetto dal menu File > New > Target;

* specificare gli intenti supportati dall'Extension scelta all'interno del file Info.plist;

* scegliere dove salvare le proprie risorse. per farlo è opportuno utilizzare un container condiviso (scelta consigliata) oppure costruire un proprio servizio in un framework privato;

* creare tante classi handler quanti sono gli intent che si vogliono gestire e definire le operazioni da svolgere al loro interno;

* eseguire i test con procedura fornita da Xcode per le applicazioni iOS.

Per quanto rigurarda l'aggiunta degli intenti anche Apple, come i suoi competitori, fa una distinzione:

* *System intents*: intenti di sistema preconfigurati da Apple;

* *Custom intents*: intenti che lo sviluppatore costruisce e personalizzare in base alle sue esigenze.

I *System intents* rappresentano le azioni più comunemente eseguite e prevedono un flusso di conversazione, opportunamente addestrato e testato, per il quale le app di sistema forniscono tutti i dati previsti.
I *Custom intents*, a differenza dei precedenti, permettono agli sviluppatori di creare

intenti personalizzati in aggiunta ai System intents. Si deve quindi definire il proprio flusso di conversazione inserendo le possibili frasi di invocazione e risposta.
In entrambe le tipologie la gestione dell'apprendimento di nuove frasi è privilegiata lato utente perché può inserirle nell'applicazione Shortcuts; tuttavia è possibile delegare questo compito alla business logic di Siri ma è una funzionalità ancora allo stato embrionale. Inoltre, qualora si utilizzassero le *Intents UI App Extension*, lo sviluppatore può implementare una grafica personalizzata senza vincoli alcuni.
Le modalità per costruire gli intenti sono riassunte nei seguenti passaggi:

* aggiungere un Intent Definition File nell'App Target;
* definire il proprio intento sulla base delle funzionalità che si vuole implementare;
* definire gli eventuali parametri (vedi sezione Gestione della comunicazione);
* aggiungere i metadati e i Siri Dialog Data alla propria conversazione;
* definire le gerarchie tra i parametri (opzionale e poco utilizzato);
* definire le eventuali e possibili shortcuts che l'utente può aggiungere dall'apposita applicazione;
* creare e settare le frasi di richiesta e risposta.

Infine Apple non fornisce delle *SDK* che permettono di realizzare quanto sopra illustrato all'interno di un progetto che possieda una propria *NLU*.

Comunicazione

Dato che lo scopo principale delle *Shortcuts* messe a disposizione da Apple non è costruire delle conversazioni, è stato deciso di non approfondire il metodo di comunicazione che utilizza. Tuttavia è noto che, come per gli altri assistenti, è possibile inserire dei parametri corrispondenti a variabili all'interno delle frasi che vengono storicizzati affinché la *Shortcuts* non venga portata a termine.

3.4.4 Proof of concept

Come *PoC* è stato pensato di costruire un'applicazione che consentisse l'esecuzione di un ordine di alimenti e quindi di aggiungere una Shortcut che inviasse l'ordine tramite Siri. Per problemi di licenze nell'account sviluppatore di Apple non è stato possibile collegare l'applicazione a Siri e quindi non è stata portata a termine con successo.

3.5 Trattamento dei dati

Per quanto concerne il trattamento dei dati scambiati durante l'esecuzione delle abilità, gli assistenti virtuali operano in modo del tutto analogo.
I dispositivi che integrano un assistente virtuale rimangono sempre in ascolto di qualsiasi parola pronunciata in modo da essere reattivi qualora venga lanciata una parola di attivazione che richiami la loro attenzione. Tuttavia solo le parole recepite dall'attivazione alla conclusione di un'abilità vengono elaborate mentre le altre non sono considerate. Questo accade perché le aziende riceverebbero una mole di dati troppo elevata per essere processata, le reti per la connessione sarebbero intasate ma soprattutto gli utenti subirebbero una violazione di privacy.

Infine per garantire sicurezza durante lo scambio dei dati utilizzano tutti il meccanismo *OAuth 2.0*[g] per l'autenticazione ed il protocollo *HTTPS* che aggiunge uno strato di crittografia.

3.6 Risultati

In questa sezione sono riportati i punti salienti dei risultati della ricerca svolta. Nella seguente tabella è rappresentato un confronto tra le funzionalità ad alto livello dei tre assistenti che sono state approfondite durante lo stage.

Funzionalità	**Assistant**	**Alexa**	**Siri**
Creazione di conversazioni personalizzate	Supporto tramite *Conversational Actions* sia con le *SDK* sia con Dialogflow.	Supporto tramite *Skill* sia con le *SDK* sia con Alexa Developer Console.	Funzionalità non supportata.
Integrazione nelle pagine Web	Supporto tramite *Content Actions*.	Funzionalità non supportata.	Funzionalità non supportata.
Integrazione nelle applicazioni	Supporto tramite *App Actions* solo su applicazioni Android.	Funzionalità non supportata.	Supporto tramite *Shortcuts* solo su applicazioni dell'ecosistema Apple.

Tabella 3.1: Tabella di confronto tra gli assistenti virtuali

Nello sviluppo di abilità mirate alle conversazioni, gli assistenti virtuali di Google e Amazon offrono rispettivamente strumenti molto simili come struttura e funzionalità mentre quello di Apple rimane più limitato.
Apple e Google inoltre rientrano nel mercato di dispositivi quali computer, smartphone e tablet e si pongono l'obiettivo di migliorare l'interazione degli utenti con i propri prodotti offrendo la possibilità di creare abilità personalizzate all'interno delle applicazioni. La differenza principale risiede nelle lingue supportate in quanto Apple fornisce il supporto a tutte le lingue in cui Siri è disponibile mentre Google per ora solo all'inglese americano.
Infine Google è l'unico ad offrire l'integrazione di Assistant nelle pagine Web per fornire all'utente una miglior interazione con il proprio motore di ricerca.
In generale ho riscontrato che gli assistenti virtuali attualmente in commercio sono pensati per un utilizzo di breve durata e perciò è molto importante progettare le loro abilità seguendo questa filosofia al fine di garantire agli utenti un'esperienza d'uso migliore.
Oltre all'analisi delle funzionalità, ho ricercato informazioni in merito all'intelligenza di Assistant, Alexa e Siri. A tal proposito ho trovato i risultati di un test che consiste nel verificare quale assistente è capace di comprendere e rispondere correttamente al maggior numero di domande su un campione di 800.

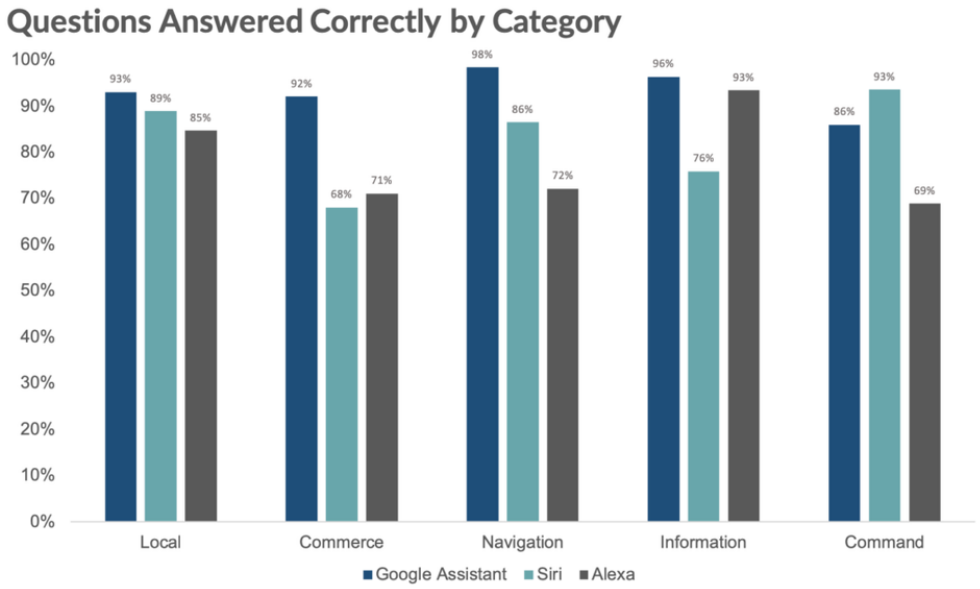

Figura 3.11: Diagramma di confronto nell'intelligenza degli assistenti

Come si può vedere dal diagramma, l'esito è nettamente a favore di Google con un punteggio mediamente al di sopra del 90% rispetto agli altri che oscillano tra 70-80%. In conclusione dall'intera ricerca è emerso come Google fornisca un insieme più completo di funzionalità per il proprio assistente e come l'intelligenza che lo caratterizza sia superiore a quella degli altri, probabilmente per la cospicua quantità di dati di varie categorie che ha a disposizione.

Capitolo 4

L'applicazione

4.1 Introduzione alle grammatiche

Zucchetti negli ultimi anni ha investito molto nella ricerca di una tecnologia che gli permettesse di interagire con i propri prodotti attraverso comandi vocali. Ora infatti possiedono delle regole per generare *grammatiche* capaci di comprendere ed elaborare un numero potenzialmente infinito di frasi del linguaggio naturale, anche se non ancora pronte per la produzione.
Gli assistenti virtuali presenti sul mercato sono basati sul seguente concetto: provare in ogni modo ad interpretare l'input ricevuto anche se non corrisponde esattamente ad uno di quelli previsti, a costo di commettere degli errori. L'obiettivo di questa filosofia è dare all'utente la percezione di utilizzare uno strumento in grado di capire e ragionare in qualsiasi condizione ed è già utilizzata in larga scala da aziende del calibro di Google, Amazon e Apple. Tuttavia, per le funzionalità offerte dalla maggior parte dei prodotti Zucchetti, tale principio non è applicabile poiché necessitano che la comprensione dell'input abbia margine di errore nullo. Un classico esempio è il trasferimento di denaro in cui, se la comprensione del comando avviene in modo errato, c'è il forte rischio di causare danni contingenti agli utenti.
Zucchetti ha perciò intrapreso una strada diversa sviluppando una tecnologia che si pone l'obiettivo di massimizzare la precisione nella comprensione dei comandi, accettando piuttosto di rigettarli qualora non abbia la piena certezza. Essa consiste in regole molto semplici ed intuitive da applicare e riassunte in cinque operazioni chiave:

* concatenazione di stringhe;
* scelta tra più stringhe;
* ripetizione di uno o più stringhe;
* opzionalità di una stringa;
* rilascio di una stringa a scelta dello sviluppatore in qualsiasi punto dell'interpretazione come segnale per l'elaborazione dei risultati.

A partire da esse viene costruita una *grammatica* che permette di interpretare un insieme finito di input rappresentanti il dominio della conversazione che si vuole intrattenere. La difficoltà principale è scegliere il corretto insieme delle frasi possibilmente pronunciabili dall'utente con l'obiettivo di ottenerne un numero elevato ma pertinente. Per fare

ciò, è necessario eseguire un'analisi probabilistica e statistica sul proprio contesto. Il problema infatti è che, qualora l'utente esprimesse una frase che differisca anche per una singola lettera da quelle generate dalla *grammatica*, non sarebbe riconosciuta.
Un esempio semplice ma dimostrativo di una *grammatica* che interpreta alcune frasi di saluto è illustrato nella seguente immagine.

```
"('salve' | 'buongiorno' | 'ciao' ['Zucchetti'] | " +
"'buon' 'pomeriggio' | " +
"'buona' ('giornata' | 'serata') | 'buonasera')"
```

Figura 4.1: Esempio di una grammatica

Nonostante il loro principio di funzionamento sia relativamente semplice da comprendere, non sono altrettanto facili da analizzare se raggiungono grandi dimensioni, soprattutto per uno sviluppatore terzo che le dovrà riutilizzare in futuro. Per migliorare questo aspetto l'azienda ha deciso di utilizzare i diagrammi *Railroad*[g] come strumento di rappresentazione.

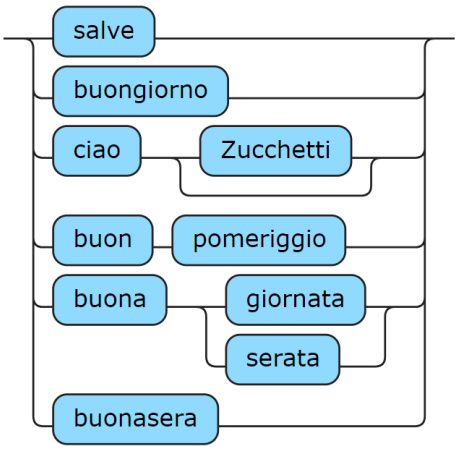

Figura 4.2: Esempio di una grammatica con railroad

Risulta evidente come questa raffigurazione della *grammatica* precedente sia molto più efficace e intuitiva.
Infine l'applicazione effettiva della *grammatica* sugli input, invece, avviene attraverso un apposito *parser*[g] che mi è stato consegnato dall'azienda per lo sviluppo del progetto.
Per riassumere gli aspetti positivi e negativi di questa tecnologia viene presentato un paragone con quella attualmente utilizzata dagli assistenti virtuali descritti in

precedenza; tuttavia non tutti sono disponibili a causa della mancanza di dati a disposizione.

Caratteristica	**Zucchetti**	**Aziende concorrenti**
Precisione nella comprensione	Estrema precisione: quando è compresa una frase si ha la certezza di averlo fatto correttamente.	Buona precisione: quando è compresa una frase si ha buone probabilità di averlo fatto correttamente ma non la certezza.
Propensione alla comprensione	Comprende solo le frasi che lo sviluppatore mette a disposizione attraverso una *grammatica*.	Cerca di interpretare anche frasi che non corrispondono esattamente a quelle a disposizione, talvolta commettendo degli errori.
Verbosità nello sviluppo	Poca verbosità in quanto, per costruzione, ad un aumento minimale vocaboli si ottiene un grande aumento delle frasi potenzialmente interpretabili.	Dati non disponibili.
Facilità della sintassi	Molto facili da gestire in quanto è basata su regole semplici che permettono di produrre molte frasi.	Dati non disponibili.
Prestazioni	Prestazioni molto elevate dovute ad un'ottima integrazione del *parser* e all'esecuzione in locale, senza quindi onere nella comunicazione.	Prestazioni altrettanto elevate con l'incognita dei tempi di latenza dovuti all'esecuzione in remoto del *parser* di interpretazione dell'input.

Tabella 4.1: Tabella di confronto tra la tecnologia Zucchetti e quella degli altri assistenti virtuali per l'interpretazione del linguaggio naturale

4.2 Analisi dei requisiti

4.2.1 Descrizione del problema

Durante l'attività di ricerca sugli assistenti virtuali, in particolare nello sviluppo del *PoC* che fa uso di Alexa, è emerso un concetto importante e caratteristico del lavoro che sta svolgendo l'azienda: la conversazionalità. Essa rappresenta la capacità di intrattenere una conversazione da parte di un software simulando la presenza di una persona.
Inoltre nella pianificazione del lavoro è inserita la costruzione di una *NLU* con relativa *grammatica* che interpreti un insieme di frasi e dia una risposta ragionata sulla base di esse.
È stato quindi deciso, in comune accordo con il tutor, di costruire un'applicazione che metta assieme la realizzazione di una propria *NLU* con capacità di conversazione finalizzata a soddisfare una determinata funzionalità e non limitata ad una coppia domanda-risposta. Il dominio dell'applicazione è simile a quello del *PoC* sviluppato con Alexa ovvero la data di nascita, solo che molto più completo.
Le frasi per cui è prevista la comprensione sono composte da:

- saluto iniziale opzionale;
- un insieme di frasi introduttive per esprimere data di nascita o di compleanno;
- insieme di espressioni per la data di nascita in qualsiasi formato o, alternativamente, per la data di compleanno, sempre in qualsiasi formato ma priva dell'anno;
- insieme di frasi per riconoscere le espressioni che definiscono il giorno di Natale;
- insieme di frasi per riconoscere le espressioni che definiscono il primo giorno di un qualsiasi mese;
- insieme di frasi per interrompere l'esecuzione in qualunque momento;
- insieme di frasi per chiedere aiuto se non si conoscessero le funzionalità offerte dell'applicazione.

4.2.2 Requisiti

Lo scopo principale è dimostrare la fattibilità di implementare la capacità conversazionale in una *NLU* costruita con la tecnologia sviluppata da Zucchetti. L'applicazione perciò si presenta sotto forma di *PoC* e non è quindi integrata in un software aziendale esistente.
Analizzando più in dettaglio gli obiettivi da raggiungere, è stata stilata una lista di requisiti obbligatori la cui fattibilità è certa. Uno tra quelli emersi, invece, è stato inserito come opzionale poiché rappresenta un miglioramento ragionevolmente non implementabile nel tempo a disposizione.
I requisiti obbligatori sono i seguenti:

1. costruzione di una *NLU* che comprenda la data di nascita espressa dall'utente, esegua un'elaborazione e prepari di conseguenza una risposta. Deve avere estrema precisione nella comprensione delle frasi anche a costo di rigettarne alcune;
2. implementazione della capacità conversazionale con relativa memoria che permetta di portare a compimento l'attività in esecuzione, provando a dare la percezione all'utente di dialogare con una persona. Più in dettaglio consiste nel richiedere i componenti mancanti della data di nascita o di compleanno oppure nella loro modifica a causa di possibili errori, affinché l'utente fornisca i dati in modo completo e corretto;
3. costruzione dell'interfaccia utente composta da:
   - interfaccia grafica minimale che permette all'utente di attivare il riconoscimento della voce;
   - interfaccia vocale completa di tutti gli accessori studiati durante l'attività di ricerca. In input risulta essere una diretta conseguenza dello sviluppo della *NLU* mentre in output è progettata sulla base delle elaborazioni prodotte.

Il requisito opzionale è il seguente:

1. generalizzazione della grammatica che permetta non solo di interpretare l'input dell'utente ma anche di generare le risposta adeguate sulla base dell'elaborazione.

4.3 Progettazione

4.3.1 NLU

La *NLU* è il nucleo di tutto l'applicativo e la sua corretta progettazione è fondamentale. Essa consiste nella creazione di una *grammatica* che interpreta l'input dell'utente sotto forma di stringhe di testo. Essa non è rappresentabile in un unico diagramma *Railroad* in quanto ha dimensioni troppo elevate, perciò sono illustrate solo le parti fondamentali. Il seguente diagramma rappresenta l'insieme di frasi per richiedere la data di nascita in cui prima viene detto il frammento che corrisponde all'introduzione del contenuto come ad esempio "sono nato il" oppure "la mia data di nascita è" e successivamente il contenuto vero e proprio ovvero giorno, mese e anno in tutti i formati esistenti.

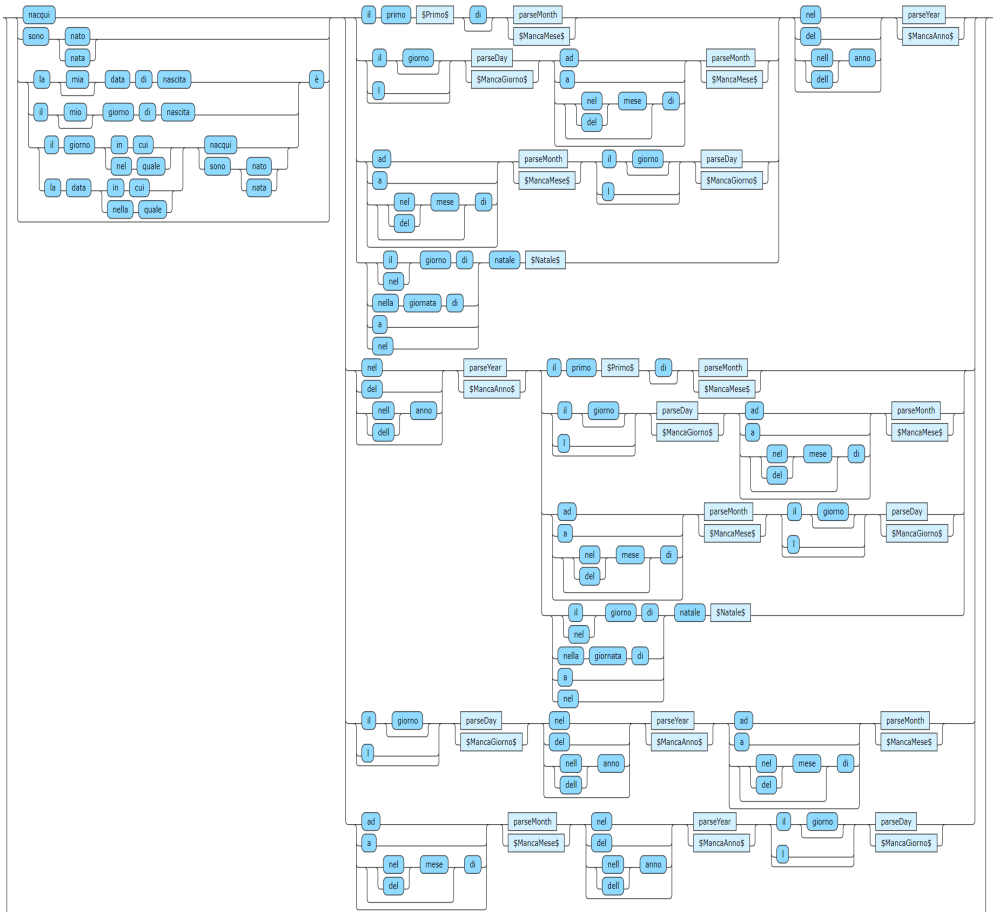

Figura 4.3: Diagramma railroad della grammatica per la data di nascita prima parte

Nella figura successiva invece è rappresentato il diagramma *Railroad* opposto in cui prima è previsto il contenuto, ovvero giorno, mese e anno in tutti i formati esistenti, ed in seguito i frammenti di frase introduttivi. Questo permette di riconoscere un maggior numero di modi in cui l'utente può esprimere la data di nascita.

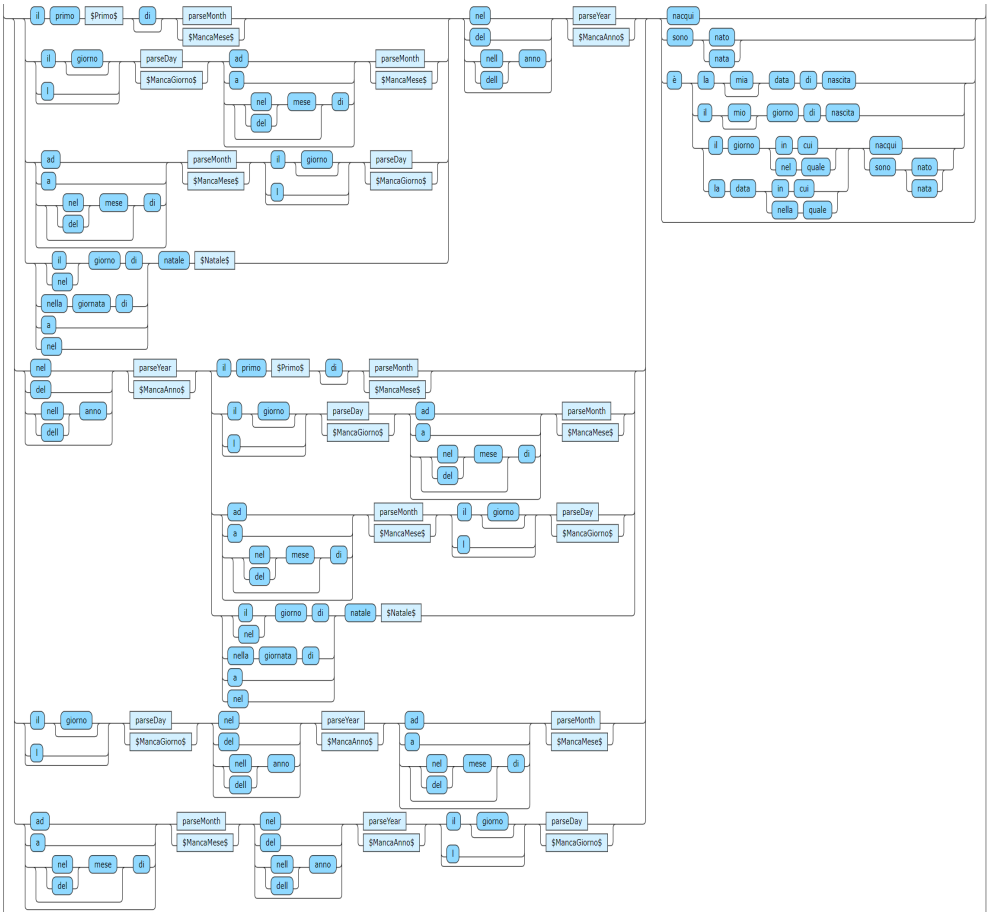

Figura 4.4: Diagramma railroad della grammatica per la data di nascita seconda parte

Le due porzioni di *grammatica* illustrate permettono globalmente di interpretare la data di nascita.
Successivamente, seguendo lo stesso principio di separazione tra i frammenti introduttivi e quelli di contesto, sono presentate le due parti di *grammatica* che illustrano la data di compleanno.
La figura seguente riporta la porzione che presenta prima la parte introduttiva e dopo quella di contenuto.

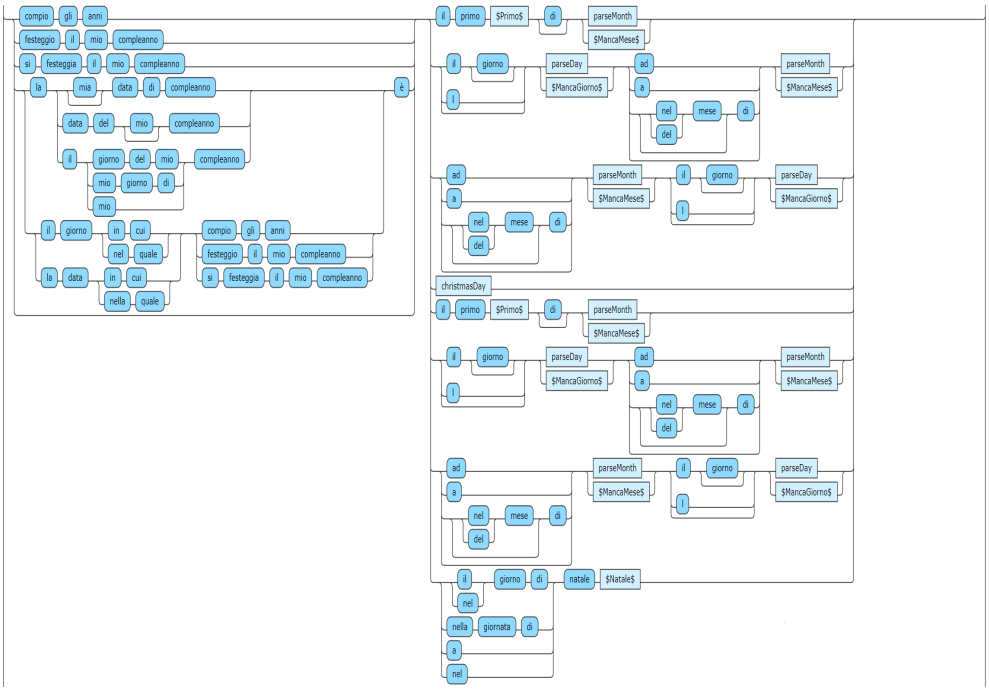

Figura 4.5: Diagramma railroad della grammatica per il compleanno prima parte

La prossima figura, invece, riporta la porzione che presenta prima la parte di contenuto e dopo quella introduttiva.

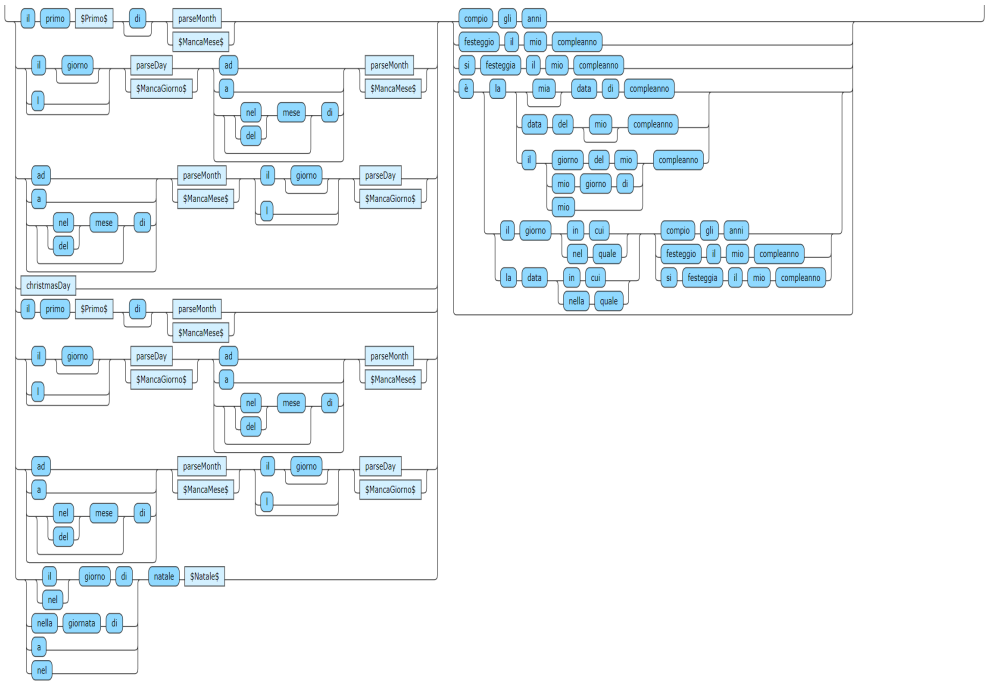

Figura 4.6: Diagramma railroad della grammatica per il compleanno seconda parte

4.3.2 Capacità conversazionale

Nella progettazione della capacità conversazionale il concetto fondamentale è la memoria. Infatti, mentre la *NLU* permette l'interpretazione del linguaggio naturale, la capacità conversazionale consente di mantenere il contesto durante l'intero dialogo.
Ho deciso di tenere traccia dei dati che lo costituiscono all'interno di un oggetto che sarà resettato ad ogni nuova conversazione. In questo modo è possibile costruire delle risposte basate sul contesto per porre domande mirate ad ottenere eventuali dati mancanti che in questo caso sono giorno, mese ed eventualmente anno.
Infine per l'implementazione della conversazione è sufficiente generare la *grammatica* descritta in precedenza in quanto è stata progettata anche per riconoscere singole parti di contenuto.
Per un ulteriore miglioramento sarebbe stato possibile tenere traccia dei dati forniti dall'utente in modo permanente, ad esempio in un database, così che ad un nuovo utilizzo, anche a distanza di tempo, il sistema si ricordasse della conversazione fatta in precedenza e potesse riprende dall'interruzione. Questo però avrebbe comportato alcuni vincoli quale, ad esempio, la realizzazione di un sistema di autenticazione che sarebbe stato troppo complesso nel tempo a disposizione.

4.3.3 Interfaccia utente

La progettazione dell'interfaccia utente si articola in due parti:

* interfaccia grafica;
* interfaccia vocale.

Interfaccia grafica

L'interfaccia grafica è stata progettata con un numero di elementi minimali ed è illustrata nella seguente immagine.

Figura 4.7: Interfaccia grafica dell'applicazione

I componenti sono un pulsante che permette all'utente di ascoltare le funzionalità fornite, un pulsante con l'immagine del microfono che attiva il riconoscimento vocale ed una casella di testo non editabile che permette di visualizzare il comando riconosciuto.

Interfaccia vocale

L'interfaccia vocale è stata progettata con l'obiettivo di garantire la migliore esperienza d'uso possibile agli utenti.
Ho utilizzato le nozioni apprese dallo studio degli assistenti virtuali e da alcuni esempi di Zucchetti per generare l'interfaccia vocale. Queste sono riportate nelle seguenti punti:

* capire la tipologia degli utenti che deve interagire con la propria applicazione. In questo caso ha avuto importanza relativa in quanto si tratta di un *PoC* e non è stata prevista una categoria specifica;
* provare a costruire esempi di dialogo molteplici verificando quali risultano più naturali su un insieme di persone sia del team di sviluppo sia di altri impieghi. Nel mio caso l'ho provato con alcuni colleghi e persone esterne;
* scegliere uno stile di conversazione che si adatti maggiormente al contesto della propria applicazione;
* dare una spiegazione iniziale di come si può interagire con l'interfaccia;
* valutare l'utilizzo di un'interfaccia grafica di ausilio per mantenere meglio il contesto, soprattutto se risulta corposo;
* gestire correttamente i possibili errori dovuti anche ad input scorretti dell'utente;
* dare la possibilità di chiedere aiuto l'utente in caso di difficoltà con frasi mirate;
* permette all'utente di interrompere l'esecuzione in qualsiasi momento senza dare la percezione di non poter uscire.

Sulla base di queste indicazioni sono stati progettati tutti i componenti espressi nell'analisi.
Gli input accettabili attraverso l'interfaccia vocale sono diretta conseguenza alla progettazione della *grammatica*, per la quale sono comunque stati applicati i principi descritti. Per l'output, invece, sono state personalizzate le risposte sulla base dell'elaborazione fornendo inoltre un set di frasi con significato uguale ma sintassi diversa, da cui viene scelta la frase a tempo di esecuzione secondo un algoritmo pseudo-casuale.

4.4 Codifica

Per realizzare l'interfaccia grafica ho implementato una pagina in *HTML* che richiama un foglio di stile in *CSS* per rendere la sua presentazione più efficace.
Per realizzare il meccanismo di riconoscimento vocale ho utilizzato un componente *jQuery* integrato in una classe Javascript, capace anche di fornire i diversi frammenti di frase che sta riconoscendo a tempo di esecuzione, mentre per la sintesi vocale ho utilizzato un oggetto della *Web Speech API*.
Per utilizzare la *NLU* ho implementato un meccanismo che simula quello degli intenti già presente negli assistenti virtuali più diffusi. Ho quindi realizzato una funzione che esegue il *parser* sulla *grammatica* al fine di verificare un'eventuale corrispondenza tra input riconosciuto e frasi a disposizione dell'applicazione. Se l'esito risulta positivo significa che è stato richiamato uno degli intenti e di conseguenza la funzione di elaborazione associata, altrimenti viene riferita all'utente la mancata comprensione. La funzione di elaborazione seziona l'input per analizzare ogni sua singola parte, aggiorna gli oggetti Javascript per la capacità conversazionale e richiama infine la funzione di costruzione della risposta da sintetizzare vocalmente con i parametri corretti.
Per realizzare la capacità conversazionale ho creato due oggetti Javascript in cui il primo contiene giorno, mese, anno e contesto che può variare tra data di nascita e di compleanno mentre il secondo contiene gli attributi booleani per giorno, mese e anno che indicano un cambiamento rispetto a quanto espresso in precedenza.

4.5 Test

L'applicazione si presenta come *PoC* e perciò non è stata prevista l'implementazione dei test di unità, integrazione, sistema e collaudo. Tuttavia si è deciso di verificare il funzionamento della *NLU* soprattutto per esplorare le loro modalità di implementazione.
Ho quindi sviluppato test caratterizzati da input sotto forma di stringa che rappresentano le frasi ragionevolmente pronunciabili dagli utenti, derivate dall'analisi statistica e probabilistica svolta durante l'attività di progettazione. Tuttavia essa potrà difficilmente essere completa di tutte le richieste: è sempre possibile che un utente, per cultura personale, si esprima in un modi diversi da quelli previsti. Perciò è importante trovare un compromesso tra tempo speso e numero di frasi verificate dall'insieme totale che la *NLU* può comprendere e scegliere degli input significativi.
Per rispondere a queste due esigenze ho quindi creato un totale di oltre 50 input che verificano tutti gli aspetti principali e gran parte di quelli marginali ma comunque importanti. In seguito viene illustrato un esempio che prende in esame alcuni dei test svolti.

```
test(birthday_grammar, 'sono nato il 9 agosto 1998')
test(birthday_grammar, 'sono nato il giorno 9 agosto 1998')
test(birthday_grammar, 'sono nato il 9 agosto del 1998')
test(birthday_grammar, 'sono nata il giorno 9 di agosto nel 1998')
test(birthday_grammar, 'sono nato il giorno 9 nel mese di agosto dell anno 1998')
test(birthday_grammar, 'compio gli anni il giorno 9 agosto')
test(birthday_grammar, 'la mia data di nascita è il 9 agosto 1987')
test(birthday_grammar, 'il mio compleanno è il 9 agosto')
test(birthday_grammar, '9 agosto')
test(birthday_grammar, '9 agosto 1998')
test(birthday_grammar, '9 1998')
test(birthday_grammar, 'agosto 1998')
test(birthday_grammar, 'il giorno 9 agosto 1998')
test(birthday_grammar, 'il 9 agosto 1998')
test(birthday_grammar, 'il giorno 9 agosto 1998 festeggio il mio compleanno')
test(birthday_grammar, 'festeggio il mio compleanno il giorno 9 agosto')
test(birthday_grammar, 'la data di nascita è il 9 agosto 1998')
test(birthday_grammar, 'il giorno del mio compleanno è il 6 luglio')
test(birthday_grammar, 'il mio giorno di compleanno è il 9 agosto')
test(birthday_grammar, 'il giorno 9 novembre è la data del mio compleanno')
test(birthday_grammar, 'il giorno 9 gennaio è il mio compleanno')
test(birthday_grammar, 'il giorno 9 marzo è il giorno del mio compleanno')
test(birthday_grammar, 'sono nato il giorno di natale del 1999')
test(birthday_grammar, 'sono nato il primo gennaio 1990')
test(birthday_grammar, 'il 9 agosto 1998 sono nato')
test(birthday_grammar, 'il giorno 9 agosto 1998 nacqui')
test(birthday_grammar, 'il 9 agosto del 1998 sono nata')
test(birthday_grammar, 'il giorno 9 di agosto nel 1998')
test(birthday_grammar, 'il giorno 28 del mese di agosto festeggio il mio compleanno')
test(birthday_grammar, 'il giorno 31 agosto compio gli anni')
test(birthday_grammar, 'il 9 agosto 1998 è la mia data di nascita')
test(birthday_grammar, 'il 9 agosto è il mio compleanno')
```

Figura 4.8: Esempio di alcune frasi di utilizzare per i test

Infine, per verificare l'efficacia dell'interfaccia vocale, ho dato la possibilità ad utenti esterni di provare l'applicazione e ciò mi ha permesso di comprendere e sistemare numerosi difetti difficilmente riscontrabili diversamente.

4.6 Risultati

I risultati ottenuti sono stati molto positivi in quanto l'applicazione funziona correttamente ed ha soddisfatto ampiamente le aspettative dell'azienda.
Dato che il nucleo principale si basa sull'interfaccia vocale, è possibile visualizzare esclusivamente il riconoscimento dei comandi e le risposte all'interno della console. A questo proposito sono riportate delle figure che illustrano un esempio di conversazione. La prima rappresenta l'attività di riconoscimento dell'input con la risposta dell'applicazione. Più in dettaglio l'utente esprime solo giorno e mese di nascita e l'applicazione gli

risponde confermando l'input e chiedendo l'anno di nascita per completare l'esecuzione.

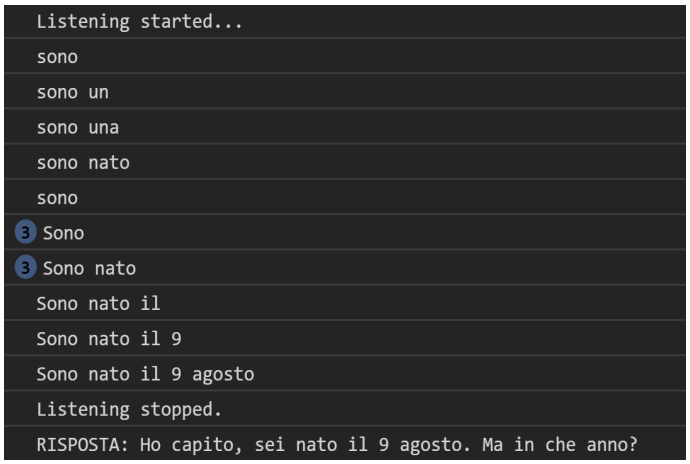

Figura 4.9: Esempio funzionamento applicazione: inizio conversazione

La seconda invece rappresenta il nuovo input con la risposta conclusiva dell'applicazione. Più in dettaglio l'utente esprime l'anno di nascita come richiesto e l'applicazione gli risponde confermando l'intera data di nascita e chiudendo la conversazione.

Figura 4.10: Esempio funzionamento applicazione: conclusione conversazione

4.7 Considerazioni

Dai risultati ottenuti sono emerse delle considerazioni legate a due tematiche:

1. ampliamento del contesto interpretabile dalla *NLU*;
2. utilizzo dei motori di regole per la costruzione di risposte;

La prima considerazione riguarda le problematiche che comporta l'ampliamento del contesto dell'applicazione verso contenuti aggiuntivi e diversi dalla data di nascita. In merito a ciò la *grammatica* rimane molto valida perché permette di ampliare

notevolmente il contesto senza troppe difficoltà ma l'interpretazione di dati aggiuntivi rischia facilmente di diventare ambigua. In particolare mi riferisco a casi come nome e cognome in cui la *NLU* fatica a distinguerli in quanto sono entrambi stringhe talvolta interscambiabili: una stringa interpretata come un nome può essere in realtà un cognome. Una soluzione non definitiva ma accettabile entro certi limiti è la mappatura di tutti i nomi e cognomi e più in generale di tutte le stringhe possibilmente ambigue. Questo però risulta fattibile solo se i domini sono piccoli, limitati e ben conosciuti.
La seconda considerazione consiste nell'utilizzo di un motore di regole per la costruzione delle componenti della risposta. In generale esso permette di creare oggetti i cui parametri sono determinati da regole preconfigurate.

Figura 4.11: Pacchetto npm del motore di regole

La sua applicazione potrebbe essere la seguente: costruire regole sulla base dei risultati ottenuti dall'elaborazione della *NLU* e della componentistica pseudo-casuale che, se saranno soddisfatte, permetteranno di costruire la risposta da ritornare all'utente sotto forma di oggetto. In questo modo rendono possibile la scrittura di codice facilmente comprensibile e manutenibile nel tempo rispetto all'utilizzo in cascata del costrutto if-else come previsto dai tutorial degli assistenti virtuali analizzati. Tuttavia questo motore di regole è ancora in via di sviluppo e si hanno a disposizione pochi esempi concreti.

Capitolo 5

Conclusione

5.1 Consuntivo finale

Gli scostamenti rilevati nelle prime tre attività sono dovuti alla difficoltà di reperimento di un computer con sistema operativo MacOS per l'analisi e la costruzione del *PoC* relativo a Siri. Perciò si è deciso di privilegiare lo studio di Assistant ed Alexa con ulteriori approfondimenti.
Nelle attività finali, invece, si sono verificati degli scostamenti perché è stato svolto un approfondimento sulla possibile utilità del motore di regole, in comune accordo con il tutor aziendale, a scapito del tempo dedicato alla documentazione.
Il consuntivo finale è quindi riportato nella seguente tabella.

Attività	**Ore pianificate**	**Ore effettive**	**Scostamento**
Studio di Assistant e implementazione di un *PoC*.	40	48	+8
Studio di Alexa e implementazione di un *PoC*.	40	48	+8
Studio di Siri e implementazione di un *PoC*.	40	32	-8
Test e documentazione comparativa di quanto svolto nelle settimane precedenti.	40	40	0
Apprendimento della tecnologia Zucchetti per il riconoscimento e l'elaborazione di comandi vocali.	40	32	-8
Realizzazione di un'applicazione che implementi una *NLU* basata su una *grammatica* costruita mediante la tecnologia di Zucchetti.	40	40	0
Implementazione della capacità conversazionale con scambio e memorizzazione di informazioni.	40	48	+8

Test e documentazione di quanto svolto nelle settimane precedenti.	40	32	-8

Tabella 5.1: Consuntivo finale

5.2 Raggiungimento obiettivi

Il raggiungimento degli obiettivi fa riferimento alla loro pianificazione descritta nella sezione §2.2.

5.2.1 Obiettivi obbligatori

* **OB-1**: obiettivo raggiunto. Inizialmente è stata svolta un'analisi preliminare di tutte le capacità di Assistant e successivamente, sulla base di indicazioni e preferenze del tutor aziendale, sono state approfondite alcune singole funzionalità;
* **OB-2**: obiettivo raggiunto. Durante l'analisi delle capacità di Assistant è stato deciso di implementare un *PoC* legato alle App Actions;
* **OB-3**: obiettivo raggiunto. Inizialmente è stata svolta un'analisi preliminare di tutte le capacità di Alexa e successivamente, sulla base di indicazioni e preferenze del tutor aziendale, sono state approfondite alcune singole funzionalità;
* **OB-4**: obiettivo raggiunto. Durante l'analisi delle capacità di Alexa è stato deciso di implementare un *PoC* legato alle Skill con capacità conversazionale;
* **OB-5**: obiettivo raggiunto. È stato redatto un documento che riporta un'analisi dettagliata e comparativa degli assistenti virtuali studiati;
* **OB-6**: obiettivo raggiunto. Sono state studiate regole e caratteristiche della tecnologia Zucchetti per costruire *grammatiche* che riconoscono il linguaggio naturale;
* **OB-7**: obiettivo raggiunto. In comune accordo con il tutor aziendale è stata realizzata un'applicazione con *NLU* propria, basata su una *grammatica* generata tramite la tecnologia Zucchetti. Il suo scopo principale è riconoscere ed elaborare la data di nascita e di compleanno dell'utente interagendo mediante un'interfaccia vocale.

5.2.2 Obiettivi desiderabili

* **OD-1**: obiettivo raggiunto. Implementazione della capacità conversazionale mirata a reperire e memorizzare tutti i dati relativi a data di nascita e di compleanno dell'utente per soddisfare lo scopo dell'applicazione.

5.2.3 Obiettivi facoltativi

* **OF-1**: obiettivo raggiunto. Inizialmente è stata fatta un'analisi preliminare di tutte le capacità di Siri e successivamente, sulla base di indicazioni e preferenze del tutor aziendale, sono state approfondite le singole funzionalità;

* **OF-2**: obiettivo non raggiunto. Durante l'analisi delle capacità di Siri è stato deciso di implementare un *PoC* che permetta l'utilizzo delle Shortcuts. Tuttavia a causa delle restrizioni dell'account sviluppatore Apple a disposizione non si è potuta portare a compimento tale attività.

5.2.4 Tabella riassuntiva

Codice	Obiettivo	Esito
OB-1	Studio e analisi delle capacità e delle modalità di utilizzo lato sviluppatore di Assistant.	Raggiunto
OB-2	Implementazione di un *PoC* che realizzi una funzionalità di Assistant accordata sulla base dei risultati della ricerca.	Raggiunto
OB-3	Studio e analisi delle capacità e delle modalità di utilizzo lato sviluppatore di Alexa.	Raggiunto
OB-4	Implementazione di un *PoC* che realizzi una funzionalità di Alexa accordata sulla base dei risultati della ricerca.	Raggiunto
OB-5	Redazione di un documento che riporta un'analisi dettagliata e comparativa degli assistenti virtuali studiati.	Raggiunto
OB-6	Studio di regole e caratteristiche dell'algoritmo di Zucchetti per la costruzione di *grammatiche* che riconoscono il linguaggio naturale.	Raggiunto
OB-7	Realizzazione di un'applicazione con una propria *NLU* basata su una *grammatica* generata mediante la tecnologia Zucchetti che interagisce con gli utenti tramite interfaccia vocale.	Raggiunto
OD-1	Implementazione della capacità conversazionale tramite lo scambio di informazioni specifiche, possibilmente memorizzate, durante la conversazione mirato a soddisfare una determinata funzionalità.	Raggiunto
OF-1	Studio e analisi delle capacità e delle modalità di utilizzo lato sviluppatore di Siri.	Raggiunto
OF-2	Implementazione di un *PoC* che realizzi una funzionalità di Siri accordata sulla base dei risultati della ricerca.	Non raggiunto

Tabella 5.2: Raggiuntimento degli obiettivi

5.3 Valutazione personale

5.3.1 Conoscenze acquisite

Durante l'attività di stage ho acquisito nuove conoscenze affrontando argomenti non presenti nel piano di studi universitario. Esse sono riportate nel seguente elenco:

* principi e funzionalità degli assistenti virtuali: ho scoperto molte funzionalità degli assistenti virtuali che ho riportato nel capitolo §3 ma soprattutto ho appreso molte nozioni sul loro funzionamento e su come possono essere utili sia agli utenti nella loro vita quotidiana sia alle azienda nella realizzazione dei loro prodotti. Le due principali sono il meccanismo degli intenti che rappresenta un nuovo modo

di gestire le interazioni con gli utenti e l'insieme dei principi di realizzazione dell'interfaccia vocale che rappresentano un'innovazione nell'interazione con gli utenti ancora poco esplorata ma ricca di potenzialità;

* comprensione del linguaggio naturale: la tecnologia Zucchetti per l'interpretazione del linguaggio naturale si basa sull'utilizzo di *grammatiche* per le quali ho avuto una formazione teorica durante il mio percorso di studi ma grazie a questa esperienza ho imparato ad applicarle in un contesto reale. Ho inoltre appreso i principi e le metodologie di realizzazione di una *NLU* e affrontato le numerose problematiche che ne derivano.

5.3.2 Competenze acquisite

Grazie all'esperienza di stage, ho anche acquisito nuove competenze che mi hanno permesso di maturare molto a livello professionale. Esse sono riportate nel seguente elenco:

* costruzione di applicazioni Android: durante lo sviluppo del *PoC* relativo ad Assistant ho imparato da autodidatta le basi della costruzione di un'applicazione Android con il linguaggio Kotlin e come attivarne le funzionalità tramite Assistant;

* costruzione di applicazioni in linguaggio Swift per l'ecosistema Apple: durante lo sviluppo del *PoC* relativo a Siri ho imparato da autodidatta le basi della costruzione di un'applicazione per iOS in linguaggio Swift e come predisporre delle *Shortcuts* personalizzabili dall'utente e attivabili tramite Siri;

* utilizzo di Javascript in ambiti nuovi: lo sviluppo dell'applicazione, a parte l'interfaccia grafica, è interamente realizzato in Javascript. Ho quindi imparato a realizzare software scritti in questo linguaggio per scopi diversi da quelli presentati durante il mio percorso di studi. Questo mi ha fatto anche capire la grande versatilità e l'ampio utilizzo che ne viene fatto nella costruzione di applicazioni;

* capacità di utilizzare nuovi strumenti: la varietà di tecnologie affrontate mi ha portato ad imparare l'utilizzo di nuovi strumenti ausiliari quali Xcode, AndroidStudio e i diagrammi *Railroad*;

* versatilità nell'utilizzo e nell'apprendimento di nuove tecnologie: la maggior parte dello stage è stato centrata su analisi e apprendimento di molte tecnologie, talvolta diverse tra loro. Ciò mi ha permesso di migliorare nell'approccio alla ricerca, nell'autoapprendimento e nella capacità di fare paragoni ricavando vantaggi e svantaggi;

* capacità di elaborare ragionamenti in ottica di innovazione: uno degli scopi principali delle mie ricerche è stato capire quali delle funzionalità e degli strumenti offerti fossero utili ai progetti aziendali. Esse infatti sono state riportate e discusse più volte con il mio tutor ed altri colleghi in sede di *brainstorming*[g]. Questo mi ha permesso di contribuire ma soprattutto imparare a fare dei ragionamenti mirati all'innovazione e al miglioramento di prodotti già esistenti o in via di sviluppo.

5.3.3 Tecnologie e strumenti utilizzati

Le tecnologie e gli strumenti con cui ho lavorato sono numerosi. Nella realizzazione dei *PoC* legati agli assistenti virtuali ho utilizzato Kotlin con Android Studio nell'applicazione per Assistant, Javascript con la console da sviluppatori nella Skill per Alexa e Swift con Xcode nell'applicazione per Siri.
Per l'applicazione che implementa la *NLU*, invece, ho utilizzato HTML e CSS per l'interfaccia grafica e Javascript versione *ES6* per tutte le altre componenti mentre come strumento ho utilizzato WebStorm.
La maggior parte di queste tecnologie non sono state affrontate, oppure solo in modo marginale, durante il mio percorso di studi; tuttavia quella di cui ho avvertito maggior carenza è indubbiamente Javascript perché ne ho fatto largo uso in ambiti totalmente nuovi.

5.3.4 Metodologia di lavoro

Lo stage si è svolto in remoto e questa è stata una nuova esperienza sia per me che per il mio tutor. Per cercare di simulare al meglio la presenza in azienda siamo rimasti in contatto quotidianamente e, in aggiunta, ho redatto un registro che riporta tutte le attività svolte in ogni giornata.
Lo svantaggio riscontrato dal lavoro in remoto è la mancanza del rapporto diretto con il tutor ed i colleghi che, a mio parere, è formativo sia dal punto vista personale che professionale. Nonostante ciò, porta con sé alcuni vantaggi quali maggior flessibilità negli orari e la non necessità di viaggiare per molto tempo per recarsi in ufficio. Inoltre, soprattutto per lavori in ambito informatico, esistono numerosi strumenti che permettono di lavorare da casa in modo efficace ed efficiente annullando così parte degli svantaggi.
Per questi motivi è stata una sfida ed una possibilità di sperimentare una nuova modalità di lavoro che mi ha permesso ugualmente di raggiungere gli obiettivi prefissati con risultati notevoli.

5.3.5 Analisi retrospettiva dei risultati

Il lavoro svolto durante lo stage non è stato strutturato per realizzare un prodotto finito e pronto all'uso ma per eseguire una ricerca esplorativa su argomenti di interesse per i progetti aziendali, che trova concretezza in alcuni *PoC* dimostrativi. In seguito ho analizzato i risultati ottenuti e da essi sono emersi alcuni importanti spunti di riflessione.
Il primo è: dall'interfaccia vocale l'utente si aspetta intelligenza. Questo l'ho percepito durante lo sviluppo dell'applicazione, quando ancora non c'era una copertura sufficiente nella comprensione dei comandi vocali. Infatti, mentre facevo eseguire dei test ad alcuni utenti esterni, spesso la *NLU* non riusciva a comprendere le frasi pronunciate facendoli spazientire; loro si aspettavano di colloquiare con un sistema intelligente al pari di una persona. Inoltre, se si considera l'aspetto conversazionale, il problema diventa ancora più accentuato in quanto l'utente si aspetta di interagire con un software che comprenda il contesto del dialogo in corso e abbia capacità di memoria.
Questa considerazione ha grande rilevanza perché evidenzia l'attenzione ai minimi dettagli che si deve prestare durante la progettazione dell'interfaccia vocale. Infatti, nonostante i grandi vantaggi che porta, quali la velocità e la comodità di utilizzo, presenta dei notevoli rischi sulla sua buona riuscita.
La seconda riflessione, invece, è la seguente: l'utilizzo dell'interfaccia vocale deve

essere giustificato rispetto a quella grafica. Questa considerazione nasce dal fatto che l'interfaccia grafica è in assoluto la più diffusa, grazie anche ai dispositivi mobili, diventando lo standard per gli utenti. Il vantaggio dell'interfaccia vocale però risiede nella maggior comodità e rapidità legata all'esecuzione di operazioni semplici per le quali risulta giustificata; tuttavia in attività complesse, possibilmente svolte in ambienti rumorosi, è assai svantaggiosa e lo sforzo per implementarla può non essere pienamente giustificato.
Dunque l'utilizzo di un'interfaccia vocale con una *NLU* rappresenta senza dubbio una tecnologia con grandi potenzialità ancora inesplorate ma dalle due precedenti riflessioni emergono i suoi limiti attuali.
La terza riflessione si propone come una possibile idea per risolvere i problemi riscontrati: costruire un'interfaccia ibrida che metta assieme gli aspetti positivi di quella vocale e di quella grafica. Più nello specifico mi riferisco alla costruzione di un'interfaccia che da un lato abbia una componente grafica per gli elementi più complessi e rilevanti, in modo che l'utente non percepisca senso di smarrimento o difficoltà nell'utilizzo e dall'altro abbia una componente vocale per le operazioni più facili e di immediata esecuzione. In questo modo si eviterebbero buona parte dei problemi espressi in precedenza senza però sfruttarne a pieno le potenzialità.
La quarta ed ultima riflessione si stacca leggermente dalle precedenti in quanto si sofferma sull'interfaccia vocale e sul possibile pattern architetturale di applicazioni che ne fanno uso. Questa riflessione è emersa in uno degli ultimi colloqui con il tutor aziendale in riferimento al pattern Model-View-Controller ed è la seguente: la View è per natura una componente passiva che esegue il rendering del modello e non possiede stato; tuttavia questo diventa impossibile da applicare con l'interfaccia vocale e la motivazione risiede nella capacità conversazionale che si vuole offrire all'utente. Più in dettaglio la View contiene sempre tutto quello che l'utente deve visualizzare mentre, durante una conversazione, il contesto si costruisce nel tempo rendendo impossibile presentare l'intero contenuto. Inoltre questo contesto, che necessariamente deve essere memorizzato, non appartiene propriamente al modello dell'applicazione in quanto non rappresenta nè i dati nè le operazioni da eseguire e nemmeno al Controller perché ha mansioni totalmente differenti. Perciò tali considerazioni portano a pensare che sia legato alla View. Da qui nasce l'idea che il pattern Model-View-Controller non è direttamente applicabile a programmi basati su interfaccia vocale ma necessita di una modifica alla View, trasformandola in un componente con capacità di memoria che mantiene il contesto della conversazione.
Queste sono le riflessioni esprimono che l'esito conclusivo dell'analisi retrospettiva dei risultati riscontrati durante l'attività lavorativa.
Per concludere, sono personalmente rimasto molto soddisfatto degli argomenti trattati, della tipologia di lavoro svolto, del rapporto lavorativo con il tutor aziendale ed in generale dell'intera esperienza di stage.

Acronimi e abbreviazioni

API Application Program Interface. 59

CSS Cascading Style Sheets. 10, 11, 46, 57

ERP Enterprise Resource Planning. 59

ES6 ECMAScript6. 10, 11, 55, 57

HTML HyperText Markup Language. 10, 11, 19, 46, 57

IBM International Business Machines Corporation. 59

NLU Natural Language Understanding. 60

PHP Hypertext Preprocessor. 11, 57

RAM Random Access Memory. 14, 57

Glossario

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica un insieme di procedure disponibili al programmatore, di solito raggruppate, che formano un set di strumenti specifici per assolvere un determinato compito all'interno di un software. Lo scopo delle *API* è ottenere un'astrazione, solitamente tra l'hardware e il programmatore o tra software a basso ed alto livello semplificando il lavoro di programmazione. 10, 23, 46

Brainstorming Il *brainstorming* è un metodo decisionale in cui la ricerca della soluzione di un dato problema è effettuata mediante sedute intensive di dibattito e confronto delle idee espresse liberamente dai partecipanti. 54, 59

Build Nello sviluppo del software, *build* indica il processo di trasformazione del codice sorgente in un artefatto eseguibile. 16, 31, 59

ERP in informatica l'*ERP, Enterprise Resource Planning* (ing. pianificazione delle risorse d'impresa), è un software di gestione che integra i processi di business rilevanti di un'azienda e le sue funzioni quali vendite, acquisti, gestione magazzino, finanza e contabilità. Integra quindi tutte le attività aziendali in un unico sistema che risulta essere indispensabile per supportare il Management. 2

Firebase *Firebase* è una piattaforma di sviluppo per applicazioni Web e mobile sviluppata da Firebase Inc. nel 2011 e acquisita da Google nel 2014. 17, 59

Grammatica Nella teoria dei linguaggi formali una *grammatica* (detta anche grammatica formale) è una struttura astratta che descrive un linguaggio (formale) in modo preciso. È definita anche come sistema di regole che delineano matematicamente un insieme potenzilamente infinito di sequenze finite di simboli appartenenti ad un alfabeto finito. 2, 6, 7, 37–39, 41, 42, 44–46, 48, 51–53, 59

HyperText Transfer Protocol In informatica *HTTP, HyperText Transfer Protocol* (ing. protocollo di trasferimento di un ipertesto) è un protocollo a livello applicativo utilizzato come principale sistema di trasmissione delle informazioni sul Web. Esiste una versione detta *HTTPS, HyperText Transfer Protocol over Secure Socket Layer* che implementa le stesse funzionalità applicando uno strato di crittografia. 15, 17, 20, 34, 59

IBM L'*IBM, International Business Machines Corporation* è la più antica azienda nel mondo dell'informatica, ha sede negli Stati Uniti ed è tra le maggiori al mondo.

Produce e commercializza hardware, software per computer, middleware e servizi informatici offrendo anche infrastrutture, servizi di hosting, cloud computing, intelligenza artificiale e consulenza nel settore informatico e strategico. 1

NLU La *NLU, Natural Language Understanding* (ing. comprensione del linguaggio naturale) è un software con intelligenza artificale capace di interpretare ed elaborare il linguaggio naturale. 2, 3, 5–7, 9, 15, 17, 18, 20, 24–26, 30, 31, 33, 39–41, 44, 46, 48, 49, 51–56

JVM La *JVM, Java Virtual Machine* è il componente della piattaforma Java che esegue i programmi tradotti in bytecode dopo una prima fase di compilazione. Alcuni dei linguaggi di programmazione traducibili in bytecode sono Java, Kotlin e Scala. 10, 60

OAuth 2.0 *OAuth 2.0* è un protocollo di rete aperto e standard, progettato specificamente per lavorare con il protocollo *HTTP*. Consente l'emissione di un token d'accesso, da parte di un server che fornisce autorizzatizioni, verso un client, previa approvazione dell'utente proprietario della risorsa cui si intende accedere. Rispetto alla sua versione precedente (*OAuth 1.0*) presenta una chiara divisione dei ruoli implementando un mediatore tra client e server. 34, 60

Parser In informatica il *parser* è un software che realizza il parsing ovvero un processo di analisi sintattica. Più in dettaglio analizza un flusso continuo di dati in input e determina la correttezza della sua struttura grazie ad una grammatica formale. 38, 39, 46, 60

PoC Con *PoC, Proof of Concept* (ing. prova di fattibilità) si intende una realizzazione incompleta e abbozzata di un determinato progetto o applicativo allo scopo di provarne la fattibilità oppure dimostrare la fondatezza di alcuni principi o concetti costituenti. Un esempio tipico è quello di un prototipo. 2, 3, 5–11, 27, 33, 39, 40, 45, 46, 51–55, 60

Railroad I *diagrammi railroad*, detti anche *diagrammi di sintassi*, consistono in una rapprensentazione grafica per grammatiche libere da contesto. 38, 41, 42, 54, 60

Software Development Kit In informatica un *SDK, Software Development Kit* (ing. pacchetto di sviluppo software), indica un insieme di strumenti per lo sviluppo e la documentazione di software. 3, 14, 17, 18, 26, 27, 33, 34, 60

Test di Turing Il *Test di Turing* è un criterio costruito dallo scienziato Alan Turing nel 1950 per determinare se una macchina sia in grado di comprendere il linguaggio naturale e più in generale di pensare. 11, 60

Bibliografia

Riferimenti bibliografici

John E. Hopcroft Rajeev Motwani, Jeffrey D. Ullman. *Automi, linguaggi e calcolabilità terza edizione*. Pearson, 2018.

Siti web consultati

Alexa developer console. URL: https://developer.amazon.com/alexa/console/ask.

Alexa: l'assistente virtuale di Amazon. URL: https://developer.amazon.com/it-IT/alexa/.

Android. URL: https://developer.android.com.

Android Studio. URL: https://developer.android.com/studio.

Articolo di confronto sull'intelligena deli assistenti. URL: https://mashable.com/article/google-assistant-dominates-siri-alexa-research/?europe=true.

Assistant: l'assistente virtuale di Google. URL: https://developers.google.com/assistant.

Diagrammi railroad o diagrammi di sintassi. URL: https://en.wikipedia.org/wiki/Syntax_diagram.

Dialogflow. URL: https://cloud.google.com/dialogflow/docs.

Linguaggio CSS. URL: https://www.w3schools.com/css/.

Linguaggio ECMAScript 6. URL: https://www.w3schools.com/Js/default.asp.

Linguaggio HTML. URL: https://www.w3schools.com/html/.

Linguaggio Kotlin. URL: https://kotlinlang.org/docs/reference/.

Siri: l'assistente virtuale di Apple. URL: https://developer.apple.com/siri/.

Validatore W3C per HTML. URL: https://validator.w3.org/.

WebStorm. URL: https://www.jetbrains.com/webstorm/.