

**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

Massimo Toffoletto: *Riconoscimento del linguaggio naturale per assistenti virtuali: interpretazione e apprendimento di comandi vocali*, Tesi di laurea triennale, © Luglio 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 riconosciute 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 esplorato ed implementato un'interfaccia vocale con un modo conversazionale di interagire che si integri con le regole già realizzate da 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 gli anni di studio.*

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

*Padova, Luglio 2020*

Massimo Toffoletto



# Indice

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduzione</b>                         | <b>1</b>  |
| 1.1      | Zucchetti S.p.A . . . . .                   | 1         |
| 1.2      | Lo stage proposto . . . . .                 | 1         |
| 1.3      | Organizzazione del testo . . . . .          | 2         |
| <b>2</b> | <b>Lo stage</b>                             | <b>3</b>  |
| 2.1      | Descrizione del progetto . . . . .          | 3         |
| 2.2      | Obiettivi . . . . .                         | 4         |
| 2.2.1    | Classificazione . . . . .                   | 4         |
| 2.2.2    | Obiettivi obbligatori . . . . .             | 4         |
| 2.2.3    | Obiettivi desiderabili . . . . .            | 4         |
| 2.2.4    | Obiettivi facoltativi . . . . .             | 5         |
| 2.3      | Pianificazione delle attività . . . . .     | 5         |
| 2.4      | Tecnologie e strumenti utilizzati . . . . . | 8         |
| 2.4.1    | Tecnologie . . . . .                        | 8         |
| 2.4.2    | Strumenti . . . . .                         | 8         |
| 2.5      | Motivazioni personali . . . . .             | 9         |
| <b>3</b> | <b>Gli assistenti virtuali</b>              | <b>11</b> |
| 3.1      | Premessa . . . . .                          | 11        |
| 3.2      | Assistant . . . . .                         | 11        |
| 3.2.1    | Introduzione . . . . .                      | 11        |
| 3.2.2    | Casi d'uso . . . . .                        | 12        |
| 3.2.3    | Conversational Actions . . . . .            | 12        |
| 3.2.4    | Content Actions . . . . .                   | 16        |
| 3.2.5    | App Actions . . . . .                       | 17        |
| 3.2.6    | Proof of concept . . . . .                  | 18        |
| 3.3      | Alexa . . . . .                             | 21        |
| 3.3.1    | Introduzione . . . . .                      | 21        |
| 3.3.2    | Casi d'uso . . . . .                        | 21        |
| 3.3.3    | Skill di conversazione . . . . .            | 21        |
| 3.3.4    | Proof of concept . . . . .                  | 25        |
| 3.4      | Siri . . . . .                              | 27        |
| 3.4.1    | Introduzione . . . . .                      | 27        |
| 3.4.2    | Casi d'uso . . . . .                        | 27        |
| 3.4.3    | Shortcuts . . . . .                         | 27        |
| 3.4.4    | Proof of concept . . . . .                  | 31        |
| 3.5      | Trattamento dei dati . . . . .              | 31        |

|          |   |           |
|----------|---|-----------|
| 3.6      | Risultati . . . . .                     | 31        |
| <b>4</b> | <b>L'applicazione</b>                   | <b>33</b> |
| 4.1      | Introduzione alle grammatiche . . . . . | 33        |
| 4.2      | Analisi dei requisiti . . . . .         | 35        |
| 4.2.1    | Descrizione del problema . . . . .      | 35        |
| 4.2.2    | Requisiti . . . . .                     | 35        |
| 4.3      | Progettazione . . . . .                 | 36        |
| 4.3.1    | NLU . . . . .                           | 36        |
| 4.3.2    | Capacità conversazionale . . . . .      | 36        |
| 4.3.3    | Interfaccia utente . . . . .            | 39        |
| 4.4      | Codifica . . . . .                      | 40        |
| 4.5      | Test . . . . .                          | 40        |
| 4.6      | Sviluppi futuri . . . . .               | 40        |
|          | <b>Glossary</b>                         | <b>41</b> |
|          | <b>Acronyms</b>                         | <b>43</b> |
|          | <b>Bibliografia</b>                     | <b>45</b> |



# Elenco delle figure

|     |  |    |
|-----|--|----|
| 1.1 | Logo di Zucchetti . . . . .  | 1  |
| 3.1 | Logo di Assistant . . . . .  | 12 |
| 3.2 | Esempio interfaccia PoC con Assistant . . . . .                          | 19 |
| 3.3 | App Actions Test Tool . . . . .  | 20 |
| 3.4 | Logo di Alexa . . . . .  | 21 |
| 3.5 | Esempio frasi RegisterBirthdayIntent . . . . .                           | 25 |
| 3.6 | Esempio JSON di risposta PoC Alexa . . . . .                             | 26 |
| 3.7 | Esempio funzionamento PoC Alexa . . . . .                                | 26 |
| 3.8 | Logo di Siri . . . . .   | 27 |
| 3.9 | Diagramma di confronto nell'intelligenza degli assistenti . . . . .      | 32 |
| 4.1 | Esempio di una grammatica . . . . .                                      | 34 |
| 4.2 | Esempio di una grammatica con railroad . . . . .                         | 34 |
| 4.3 | Diagramma railroad della grammatica per la data di nascita prima parte   | 37 |
| 4.4 | Diagramma railroad della grammatica per la data di nascita seconda parte | 38 |
| 4.5 | Diagramma railroad della grammatica per il compleanno prima parte .      | 38 |
| 4.6 | Diagramma railroad della grammatica per il compleanno seconda parte      | 39 |
| 4.7 | Interfaccia grafica dell'applicazione . . . . .                          | 39 |

# Elenco delle tabelle

|     |  |    |
|-----|--|----|
| 2.1 | Pianificazione delle attività . . . . .                    | 5  |
| 3.1 | Tabella di confronto tra gli assistenti virtuali . . . . . | 32 |



# Capitolo 1

## Introduzione

### 1.1 Zucchetti S.p.A

Zucchetti è 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 in cui risiede 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 avuto un successo sempre maggiore tanto da ottenere collaborazioni con aziende del calibro di *IBM*<sup>[g]</sup>. A partire da questo, Zucchetti ha continuato a perseguire l'innovazione del proprio prodotto integrandolo con nuovi moduli quali *ERP*<sup>[g]</sup> e la più recente fatturazione elettronica, con l'obiettivo di conferire maggiore flessibilità e adattabilità per ogni tipologia di impresa, senza limitarsi ai commercialisti. Forte di questo, Zucchetti si è espansa a livello nazionale ed internazionale ponendosi sul mercato con una vasta gamma di servizi per numerosi settori quali industrie manifatturiere, trasporti, logistica, sanità, fitness e molti altri.

### 1.2 Lo stage proposto

Uno dei pilastri di Zucchetti che ha contribuito alla sua espansione è l'innovazione e la pensione 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. Il coordinatore di tale reparto è il dott. Gregorio Piccoli che mi ha proposto il progetto e offrendosi come mio tutor.

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. L'obiettivo è quindi implementare un'interfaccia diversa da quella grafica, con caratteristiche proprie, che esprima un nuovo modo di comunicare con le applicazioni; seppur ancora poco sviluppato ha delle grandi potenzialità.

Per raggiungere tale obiettivo 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 capaci di riconoscere comandi vocali. Tuttavia è una tecnologia ancora in fase di sviluppo ed in merito a ciò mi sono stati proposte due tematiche come progetto di stage, 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 la possibilità di utilizzarli per permettere agli utenti comunicare con il software gestionale dell'azienda;
2. esplorare la possibilità di implementare la capacità conversazionale in una *NLU*<sup>[g]</sup> che utilizzi una grammatica costruita con la nuova tecnologia dell'azienda, ispirandosi anche agli assistenti virtuali precedentemente studiati.

### 1.3 Organizzazione del testo

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

**Il terzo capitolo** descrive il lavoro di ricerca svolto sugli assistenti virtuali e le considerazioni tratte al termine.

Riguardo la stesura del testo, relativamente al 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*<sup>[g]</sup>;
- \* 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.

Nel corso degli ultimi anni, l'azienda sta affrontando la sfida della comprensione di comandi vocali per permettere agli utenti di comunicare 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 si è costruito il progetto di stage che si articola in due parti:

1. studio degli assistenti virtuali più comuni ed utilizzati;
2. creazione di un'applicazione sotto forma di  $PoC^{[g]}$  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 principalmente le loro capacità conversazionali e di interazione con le applicazioni, prestando particolare attenzione alla costruzione di abilità personalizzate e integrate nei propri prodotti che comunichino con gli assistenti ma che posseggano una propria *NLU* da utilizzare. Le altre capacità suscitano un interesse minore in quanto l'azienda punta alla realizzazione di un assistente utile solo nei propri prodotti e non su larga scala. Inoltre, per dare concretezza alle funzionalità più significative che sono state riscontrate, è richiesto un *PoC* per ogni assistente.

Sulla base dei risultati ottenuti si concretizza la seconda parte del lavoro: creare un'applicazione con un'interfaccia vocale e una propria *NLU* basata su grammatiche costruite mediante la tecnologia di Zucchetti, che permetta di intrattenere una conversazione con l'utente.

Il contenuto dell'applicazione non è vincolato dall'azienda in quanto l'obiettivo è approfondire l'abilità conversazionale e non costruire un'applicazione direttamente utilizzabile

nei software Zucchetti. In comune accordo è stato quindi scelto il contesto della data di nascita poiché è ricco di sfumature sia nelle richieste che nelle risposte e possiede parametri specifici e funzionali alla sperimentazione sulla capacità di conversazione. Dunque lo scopo generale del progetto di stage è svolgere un'attività di ricerca e di approfondimento nella tematica del linguaggio naturale poiché è una tecnologia non ancora pienamente affermata ma con ampi margini di sviluppo.

## 2.2 Obiettivi

### 2.2.1 Classificazione

La classificazione degli obiettivi permette di identificarli univocamente e 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 riconoscibile 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 implementi 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 implementi una funzionalità di Alexa accordata sulla base dei risultati della ricerca;
- \* **OB-5**: redazione di un documento che tratti l'analisi dettagliata e comparativa in merito alle tecnologie degli assistenti virtuali studiati;
- \* **OB-6**: studio di regole e caratteristiche dell'algoritmo di Zucchetti per la costruzione di grammatiche che riconoscono il linguaggio naturale;
- \* **OB-7**: realizzazione di un'applicazione con una propria *NLU* basata su una grammatica generata dall'algoritmo di Zucchetti e con capacità conversazionale 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:

| Durata in ore | Date (inizio - fine)    | Attività   |
|---------------|-------------------------|--|
| 40            | 04/05/2020 - 08/05/2020 | Studio di Assistant e implementazione di un <i>PoC</i> .   |
| 40            | 11/05/2020 - 15/05/2020 | Studio di Alexa e implementazione di un <i>PoC</i> .   |
| 40            | 18/05/2020 - 22/05/2020 | Studio di Siri e implementazione di un <i>PoC</i> .  |
| 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 di sistema creato da Zucchetti per l'interazione mediante comandi vocali.                          |
| 40            | 08/06/2020 - 12/06/2020 | Realizzazione di un'applicazione che implementi una grammatica costruita utilizzando la tecnologia di Zucchetti. |
| 40            | 15/06/2020 - 19/06/2020 | Implementazione del meccanismo per lo scambio di informazioni con capacità di memorizzazione.                    |
| 40            | 22/06/2020 - 26/06/2020 | Test e documentazione di quanto svolto nelle settimane precedenti.   |

Tabella 2.1: Pianificazione delle attività

- \* **OD-1:** implementazione di un meccanismo che permetta lo scambio di informazioni specifiche e possibilmente memorizzate durante la conversazione.

### 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 implementi 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 stati stabiliti sia una chiamata al termine di ogni giornata che 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. 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 relativi alle abilità;
- \* progettazione delle abilità con attenzione all'interfaccia vocale;
- \* interazione dell'utente 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 relativi alle abilità;
- \* progettazione delle abilità con attenzione all'interfaccia vocale;
- \* interazione dell'utente 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 relativi alle abilità;
- \* progettazione delle abilità con attenzione all'interfaccia vocale;
- \* interazione dell'utente 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 mirata a riportare 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.

### 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*<sup>[g]</sup>, comprese quelle Android. 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 *ECMAScript6 (ES6)* è lo standard di javascript, un linguaggio di programmazione debolmente tipizzato, orientato agli oggetti e agli eventi che viene utilizzato nella programmazione Web lato client o lato server tramite Node.js. 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 di 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

*HyperText Markup Language (HTML)*<sup>[g]</sup> 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 e veloce nella sintassi e non necessito di piena compatibilità con tutti i browser o le versioni meno recenti.

#### CSS

*Cascading Style Sheets (CSS)*<sup>[g]</sup> 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 minimale necessaria.

### 2.4.2 Strumenti

#### Android Studio

Android Studio è un ambiente di sviluppo integrato per realizzare applicazioni da utilizzare nella piattaforma Android e 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 di *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 nell'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 *Hypertext Preprocessor (PHP)*<sup>[g]</sup>. Durante il mio lavoro è stato ausiliario per l'implementazione dell'intera applicazione 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.

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*<sup>[g]</sup>. È un argomento non trattato nel percorso di studi che ho svolto ma a mio parere molto interessante.

La seconda motivazione risiede sul contenuto specifico del progetto che ho personalmente trovato molto stimolante perché mi ha permesso di imparare e realizzare qualcosa di innovativo ma in gran parte basato su conoscenze apprese durante l'attività di ricerca. 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 e 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 ma, nonostante siano software di tre aziende diverse, il loro funzionamento è molto simile: lo sviluppatore deve costruire un'abilità, addestrarla 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 meccanismo di funzionamento delle abilità trattate: gli intenti. Essi consistono in un'azione che soddisfa una richiesta vocale effettuata da un utente e il servizio che la esegue, più concretamente il codice, viene detto *fulfillment*. La seconda proprietà in comune è la possibilità di integrare l'interfaccia vocale a quella grafica ove possibile. Questo non viene trattato nell'analisi ma permetterà di giungere a conclusioni interessanti nel seguito.

L'obiettivo dell'analisi svolta è presentare le sole funzionalità degli assistenti che sono state studiate, fornendo principi e metodi ad alto livello relative alla progettazione e all'implementazione ma non nel dettaglio perché spiegate nelle pagine di documentazione.

### 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.

Reso pubblico dal 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 *Random Access Memory (RAM)*<sup>[8]</sup> 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 invocare Siri. Infine è fruibile nei



**Figura 3.1:** Logo di Assistant

dispositivi della linea Home e Nest di Google, costruiti e pensati appositamente per ottimizzarne le funzionalità.

### 3.2.2 Casi d'uso

- \* 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 esperienza di conversazione personalizzata con gli utenti. Esse infatti permettono di gestire le richieste rivolte all'Assistente e le risposte costruite 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 alle apposite *Software Development Kit*<sup>[g]</sup>.

#### Funzionamento

Il principio di funzionamento si articola nei seguenti passi:

1. un utente lancia una richiesta sotto forma di comando vocale al dispositivo che ospita Assistant;
2. il dispositivo esegue la porzione di software che riconosce il comando vocale trasformandolo in stringhe di testo;
3. il dispositivo invia la stringa riconosciuta ad un server remoto di Google, tramite protocollo *HyperText Transfer Protocol*<sup>[g]</sup>, dove risiede la *NLU* per l'elaborazione;
4. la *NLU* del server remoto prova a verificare 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 associato con il suo servizio di *fulfillment* che si occupa dell'esecuzione;
7. la conclusione dell'esecuzione prevede la costruzione e l'invio della risposta al dispositivo mittente;
8. il dispositivo riferisce la risposta 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.

Qualora la richiesta dell'utente fosse 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. L'attività successiva, invece, è la progettazione che deve essere 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*<sup>[6]</sup> della *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: l'utente può pronunciare parole aggiuntive 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 direttamente l'intento da eseguire. In questo caso la business logic di Assistant 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*<sup>[g]</sup> è quello predefinito.

Per costruire una *Conversational Action* necessita un agente ovvero un modulo di comprensione del linguaggio naturale che gestisce le conversazioni con gli utenti sgravando lo sviluppatore di 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 uno strumento simile con Alexa; tuttavia i principi di funzionamento e implementazione sono molto simili.

Le Conversational Actions SDK consistono in un'interfaccia *HyperText Transfer ProtocolS* che permette di costruire ed eseguire le *Conversational Actions* all'interno di una propria applicazione per elaborare le richieste effettuate ad Assistant. Grazie all'interfaccia *HyperText Transfer ProtocolS* infatti Assistant può comunicare con applicazioni terze; 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 *Software Development Kit* 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 *Software Development Kit* sono uguali:

- \* Default Actions: azione cui corrisponde l'evento chiamato *GOOGLE\_ASSISTANT\_WELCOME* per Dialogflow e *actions.intent.MAIN* per le *Software Development Kit* 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;

- \* Additional Actions: azione aggiuntiva alla Default Actions utilizzata per aggiungere e specificare le capacità dell'*Action*. 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. 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 genera una risposta.

Qui emerge una grande differenza tra Dialogflow e *Software Development Kit*: 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 della proprio applicativo, diretta conseguenza dell'utilizzo delle *Software Development Kit*, 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 prova e, qualora sia utilizzato, vieta lo scambio di denaro;
- \* 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 richiedere determinati dati possibilmente importanti per la corretta esecuzione dell'*Action* 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, Assistant utilizza delle variabili all'interno delle frasi di richiesta dette *tips*. Essi devono essere definiti 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 un database.

### 3.2.4 Content Actions

#### Descrizione

Le *Content Actions* sono delle funzionalità basate su Assistant ma integrate nelle pagine Web. In particolare, inserendo dati strutturati all'interno di una pagina Web, permettono di costruire *Actions* che ne presentano il contenuto in modo interattivo ad una richiesta vocale di un utente.

#### Funzionamento

Il loro funzionamento verte esclusivamente sull'algoritmo di Google ed è 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 *Actions* stessa.

#### Dati strutturati

Le *Content Actions* si basano sui dati strutturati. Essi sono rappresentati da 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 multimediale, 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 che ne fanno uso.

#### 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 podcast 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 da Google.

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 le 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 composto dai seguenti passi:

1. un utente invoca una *App Actions* attraverso un apposito comando vocale;
2. il dispositivo esegue il software che riconosce il comando vocale trasformandolo in stringhe di testo;
3. il dispositivo invia la stringa riconosciuta ad un server remoto di Google, tramite protocollo *HyperText Transfer ProtocolS*, dove risiede la *NLU* per l'elaborazione;
4. la *NLU* del server remoto prova a verificare 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 ritornando 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 *App Actions* presuppone l'esistenza di un'applicazione costruita per Android a cui aggiungerla. Gli elementi cardine la decisione della frase di invocazione in quanto non è possibile averne solo una per ogni intento e 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 tra quelli possibili. Gli intenti disponibili sono solo quelli preconfigurati da Google e i macro argomenti che trattano sono:

- \* attivare funzionalità delle applicazioni;
- \* effettuare degli ordini online;

- \* operazioni in ambito finanziario;
- \* ordinare cibi e bevande da un menù preconfigurato;
- \* monitoraggio di salute e fitness;
- \* richiedere trasporto tramite Taxi.

Ad oggi non è ancora possibile costruire un intento personalizzato.

### Implementazione

Dopo aver progettato la *Action* si passa all'implementazione. Questa 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 altrimenti non è possibile richiamarla.

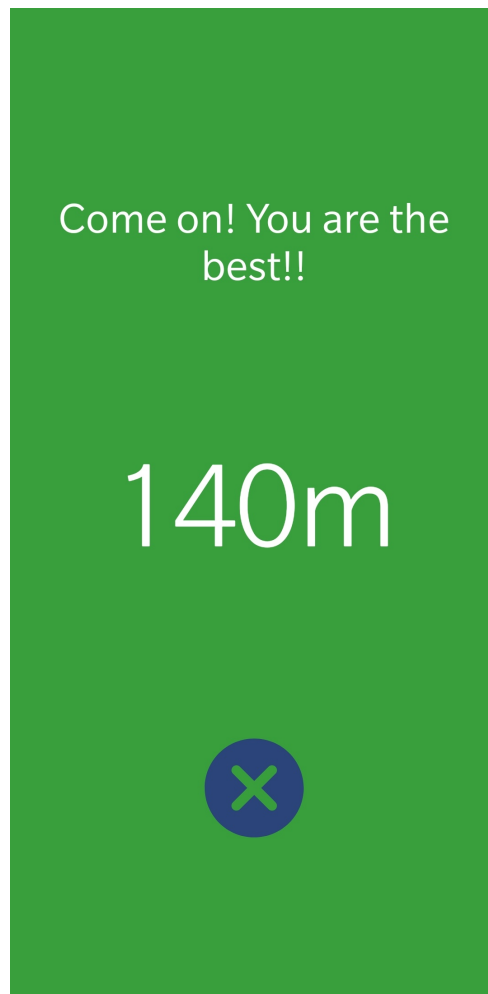
## 3.2.6 Proof of concept

### 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* sull'argomento del fitness. Il suo scopo è verificare la fattibilità e le potenzialità delle *App Action*, nonostante il loro attuale supporto alla sola lingua inglese, con un esempio concreto. Ho quindi deciso di realizzare un timer con un'interfaccia grafica minimale che calcoli il tempo che si impiega nel svolgere attività fisica per ogni sport disponibile nei *built-in intents*.

### Implementazione

Nell'attività di implementazione ho seguito le linee guida fornite da Google e come primo compito ho realizzato un'applicazione Android che presenta tutte le funzionalità previste e fruibili tramite l'interfaccia grafica. Più in dettaglio l'utente può attivare il timer dall'apposito pulsante, fermare l'attività in qualunque momento e visualizzare i risultati nello storico.



**Figura 3.2:** Esempio interfaccia PoC con Assistant

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.

### Test

Per eseguire 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.

Edit saved preview ?

Delete PreviewUpdate Preview

Test saved preview

1) Configure ?

actions.intent.START\_EXERCISE

exercise:

```
{
  "@type": "Exercise",
  "name": "Running",
  "@context": "http://schema.googleapis.com"
}
```

2) Select Target Device

ONEPLUS A6003 (ffc56a17) - ONLINE

3) Test App Action ?

Localeen-US

Invocation namefirstApp

ADB commandadb -s ffc56a17 shell am start -a android.inteRunCopy

**Figura 3.3:** 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 3.4:** 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 Echo di Amazon 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 analizzato 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 *Application Program Interface* 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 danno la possibilità di 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. il dispositivo attiva un apposito elaboratore che riconosce le parole pronunciate trasformandole in stringhe di testo;
3. il dispositivo manda la stringa riconosciuta ad un server remoto per l'elaborazione;

4. il server remoto attiva la propria *NLU* che prova a cercare una 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 positivo viene selezionato l'intento;
6. prima di eseguire il codice dell'intento vengono invocati gli eventuali *Request Interceptors* definiti dallo sviluppatore, illustrati in un paragrafo successivo;
7. viene richiamato gestore dell'intento, rappresentante il codice da eseguire, che porterà a termine l'intento;
8. dopo aver gestito l'evento, vengono richiamati gli eventuali *Response Interceptors* definiti dallo sviluppatore, illustrati in un paragrafo successivo;
9. viene costruita la risposta e ritornata al dispositivo che ospita l'assistente;
10. il dispositivo riferisce la risposta 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. 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. Durante l'esecuzione della *build* della *Skill*, Alexa sarà addestrato sulle frasi immesse per interpretarle correttamente.

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: l'utente può pronunciare parole aggiuntive 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. In merito a ciò Amazon offre due strumenti:

- \* Alexa Developer Console;
- \* Alexa SDK.

Il primo è Alexa Developer Console è uno strumento che integra Alexa Skills Kit e fornisce un'interfaccia allo sviluppatore 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 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 formalizzare ed inserire nell'interfaccia grafica le possibili frasi di invocazione, richiesta e risposta e gli eventuali slot. Con definizione degli intenti, invece, si intende formalizzare le azioni che la *Skill* deve essere capace di eseguire. In particolare è possibile scegliere tra gli intenti *built-in* ovvero quelli già preconfigurati da Amazon e quelli personalizzati, interamente a carico dello sviluppatore, dei quali è necessario definire:

- \* 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 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*. È disponibile nei linguaggi Javascript/Type-script, Java e Python. 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 per lo scambio di dati tra Alexa ed il proprio applicativo che avviene sotto forma di oggetti JSON.

### 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 *Software Development Kit*, 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 per 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 riguardanti 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 in quanto 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*. Essi devono essere definiti 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 un database.

### 3.3.4 Proof of concept

#### Analisi dei requisiti

Per il proof of concept 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 oltre all'analisi teorica. 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 e che maschera la complessità della *Software Development Kit* e della comunicazione con essa. 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.

|  |
|--|
| sono nato a {month}                                |
| sono nato il {day}                                 |
| il mio compleanno è il {day} di {month} del {year} |
| sono nato il {day} {month} del {year}              |

**Figura 3.5:** 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 3.6:** 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 3.7:** 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 3.8:** 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 analizzato 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 possibile accedere all'applicazione *Shortcuts* (comandi se le impostazioni sono in lingua italiana) e al suo 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. il dispositivo attiva un apposito elaboratore che riconosce le parole pronunciate trasformandole in stringhe di testo;

3. il dispositivo manda la stringa riconosciuta ad un server remoto per l'elaborazione;
4. La *NLU* di Siri presente nel server remoto prova a cercare una 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 della corrispondenza ha dato esito positivo viene selezionato l'intento;
6. viene richiamata la porzione di codice che porterà a termine l'intento;
7. viene costruita e ritornata la risposta al dispositivo che ospita l'assistente;
8. il dispositivo riferisce la risposta all'utente che potrà procedere con una nuova richiesta fino al termine dell'esecuzione dell'abilità.

### Progettazione

Nella costruzione di una *Shortcut* la prima attività da svolgere è l'analisi dei requisiti ovvero comprendere dettagliatamente il comportamento che si vuole ottenere. 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 e di conseguenza verrà aggiornata *Shortcuts*.

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: l'utente può pronunciare parole aggiuntive 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 la Shortcut o l'intento da eseguire. In questo caso la business logic di Siri 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 e 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 e sia per l'utilizzo dei propri strumenti ma anche 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 l'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 riguarda 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 degli intenti personalizzati in aggiunta ai System intents. Si deve quindi definire il proprio flusso di conversazione inserendo le possibili frasi di invocazione e di risposta. In entrambi 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 *Software Development Kit* che permettano di realizzare quanto illustrato all'interno di un progetto che possieda una propria *Software Development Kit*.



### Comunicazione

Dato che lo scopo principale delle *Shortcuts* che Apple mette a disposizione 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 l'applicazione.

### 3.5 Trattamento dei dati

Per quanto concerne il trattamento dei dati che vengono scambiati durante l'esecuzione delle abilità, gli assistenti virtuali operano in modo del tutto simile.

Tutti 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 il meccanismo *OAuth 2.0*<sup>[5]</sup> per l'autenticazione ed il protocollo *HyperText Transfer ProtocolS* che aggiunge uno strato di crittografia.

### 3.6 Risultati

In questa sezione vengono riportati i risultati salienti della ricerca svolta. Nella seguente tabella è presentato un confronto tra le funzionalità ad alto livello dei tre assistenti che sono state più rilevanti per lo stage. Nello sviluppo di abilità mirate alle conversazioni gli assistenti virtuali di Google e Amazon offrono rispettivamente strumenti molto simili come struttura e funzionalità offerte.

Apple e Google 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 proprie 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 in inglese.

Infine Google è l'unico ad offrire l'integrazione di Assistant nell'interazione con le pagine Web.

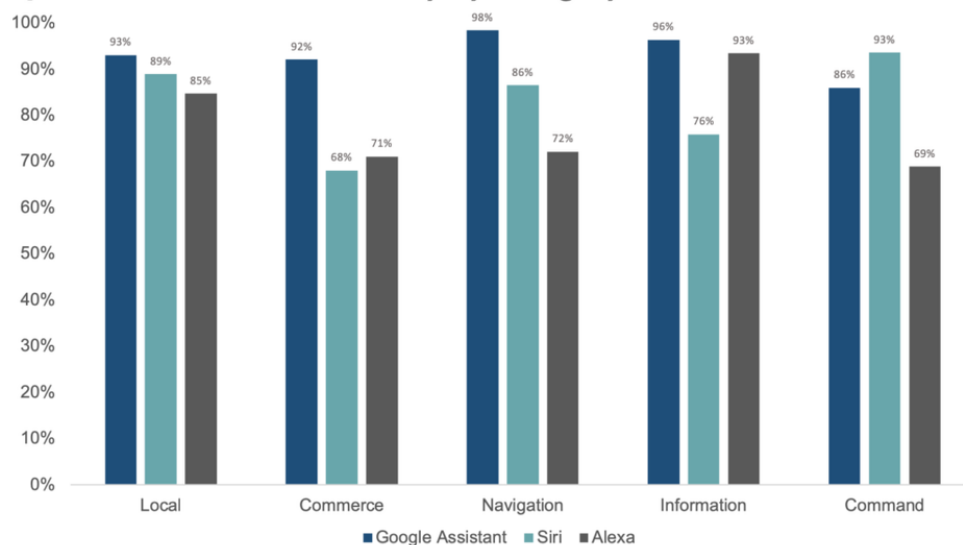
In generale ho riscontrato che gli assistenti virtuali attuali sono pensati per un utilizzo di breve durata perciò è molto importante progettare le loro abilità seguendo questo principio per dare agli utenti un'esperienza d'uso migliore. Oltre all'analisi delle funzionalità ho analizzato l'intelligenza degli assistenti e ho trovato i risultati di un

| Funzionalità                              | Assistant   | Alexa   | Siri  |
|---|---|---|---|
| Creazione di conversazioni personalizzate | Supporto tramite <i>Conversational Actions</i> sia con le <i>Software Development Kit</i> sia con Dialogflow. | Supporto tramite <i>Skill</i> sia con le <i>Software Development Kit</i> sia con Alexa Developer Console. | Funzionalità non supportata.  |
| Integrazione nelle pagine Web             | Supporto tramite <i>Content Actions</i> .   | Funzionalità non supportata.  | Funzionalità non supportata.  |
| Integrazione nelle applicazioni           | Supporto tramite <i>App Actions</i> solo su applicazioni Android.   | Funzionalità non supportata.  | Supporto tramite <i>Shortcuts</i> solo su applicazioni dell'ecosistema Apple. |

**Tabella 3.1:** Tabella di confronto tra gli assistenti virtuali

test svolto su di essi in questo ambito che consiste nel verificare quale assistente è in grado di comprendere e rispondere correttamente al maggior numero di domande su un campione di 800.

### Questions Answered Correctly by Category



**Figura 3.9:** Diagramma di confronto nell'intelligenza degli assistenti

Come si può vedere dal diagramma, l'esito è stato a favore di Google con un netto vantaggio.

Dunque da questa 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 ed ora sta realizzando delle regole che consentono di generare un numero di *grammatiche* potenzialmente infinito e capace di comprendere ed elaborare il linguaggio naturale. Gli assistenti virtuali presenti sul mercato sono basati sul seguente concetto: provare ad interpretare l'input ricevuto anche se non corrisponde esattamente ad uno di quelli previsti a costo di commettere degli errori. Questa filosofia è mirata a dare all'utente la percezione di utilizzare uno strumento in grado di capire e ragionare in qualsiasi momento 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 rischio di causare danni contingenti agli utenti.

Zucchetti ha perciò intrapreso una strada diversa sviluppando una tecnologia con l'obiettivo di massimizzare la precisione della comprensione dei comandi, accettando piuttosto di rigettarli. 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;
- \* rilasciare una stringa a scelta dello sviluppatore in qualsiasi punto all'interno come segnale per l'elaborazione.

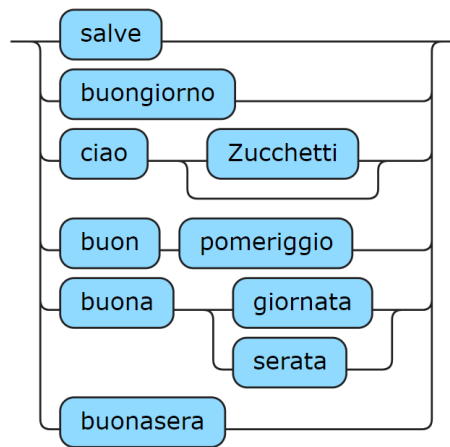
A partire da esse viene costruita una *grammatica* che permette di interpretare un insieme finito di input che rappresentano il dominio della conversazione che si vuole intrattenere. La difficoltà principale è scegliere il corretto insieme delle possibili frasi che l'utente può pronunciare con l'obiettivo di ottenerne un numero molto elevato ma pertinente, eseguendo 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 permette di interpretare 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 interpretare se raggiungono grandi dimensioni, soprattutto per uno sviluppatore terzo che le dovrà riutilizzare in futuro. Per migliorare questo aspetto l'azienda ha quindi deciso di utilizzare i diagrammi *railroad*<sup>[g]</sup> come strumento di rappresentazione come si può vedere nella figura successiva.



**Figura 4.2:** Esempio di una grammatica con railroad

Risulta evidente infatti come questa raffigurazione sia molto più efficace e intuitiva. Inoltre per l'applicazione effettiva della *grammatica* sugli input dell'utente avviene attraverso un apposito *parser*<sup>[g]</sup> che mi è stato consegnato dall'azienda per lo sviluppo del progetto.

Infine per riassumere gli aspetti positivi e negativi di questa tecnologia viene riportata la seguente tabella.

## 4.2 Analisi dei requisiti

### 4.2.1 Descrizione del problema

Durante l'attività di ricerca ed analisi sugli assistenti virtuali, in particolare nello sviluppo del *PoC* che fa uso di Alexa, è emerso un concetto importante, caratteristico anche 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 di lavoro è simile a quello del *PoC* sviluppato con Alexa ovvero la data di nascita, solo che molto più completo. Le frasi pronunciabili dagli utenti per cui è prevista la comprensione sono composte da:

- \* saluto iniziale opzionale;
- \* un insieme di frasi introduttive per esprimere la data di nascita nel formato giorno, mese e anno o, alternativamente, la data di compleanno nel formato giorno, mese;
- \* insieme di espressioni per la data di nascita e conseguentemente del sottoinsieme data di compleanno;
- \* insieme di frasi per riconoscere come data il giorno di Natale;
- \* insieme di frasi per riconoscere come data il primo giorno di un qualsiasi mese;
- \* insieme di frasi per interrompere in qualunque momento l'esecuzione;
- \* insieme di frasi per chiedere un eventuale aiuto sulle funzionalità 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 dell'interfaccia utente:
  - \* interfaccia grafica minimale che permetta 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.

2. costruzione di una *NLU* che comprenda la data di nascita espressa dall'utente, esegua un'elaborazione e prepari una risposta adatta. Deve avere estrema precisione nella comprensione delle le frasi anche a costo di rigettarne alcune;
3. implementazione della capacità conversazionale con relativa memoria che permetta di portare a compimento l'attività in esecuzione, dando maggiormente 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 affinché l'utente li fornisca in modo completo.

Il requisito opzionale è il seguente:

- \* generalizzazione della grammatica in modo che permetta non solo di interpretare l'input dell'utente ma anche di generare la risposta 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 *grammatica* generata dallo sviluppatore che interpreta l'input dell'utente e nella sua applicazione alle stringhe di testo. La *grammatica* non è rappresentabile in un unico diagramma in quanto ha dimensioni troppo elevate perciò sono illustrate le parti fondamentali.

Il seguente diagramma rappresenta l'insieme di frasi per richiedere la data di nascita in cui prima viene detto il frammento di frase 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 tutte le possibili combinazioni.

Nella figura successiva invece è rappresentato il diagramma *railroad* opposto in quanto prima prevede il contenuto ovvero giorno, mese e anno sempre in tutte le possibili combinazioni ed in seguito le sue frasi introduttive. Questo permette di riconoscere più versioni in cui l'utente può esprimere la data di nascita.

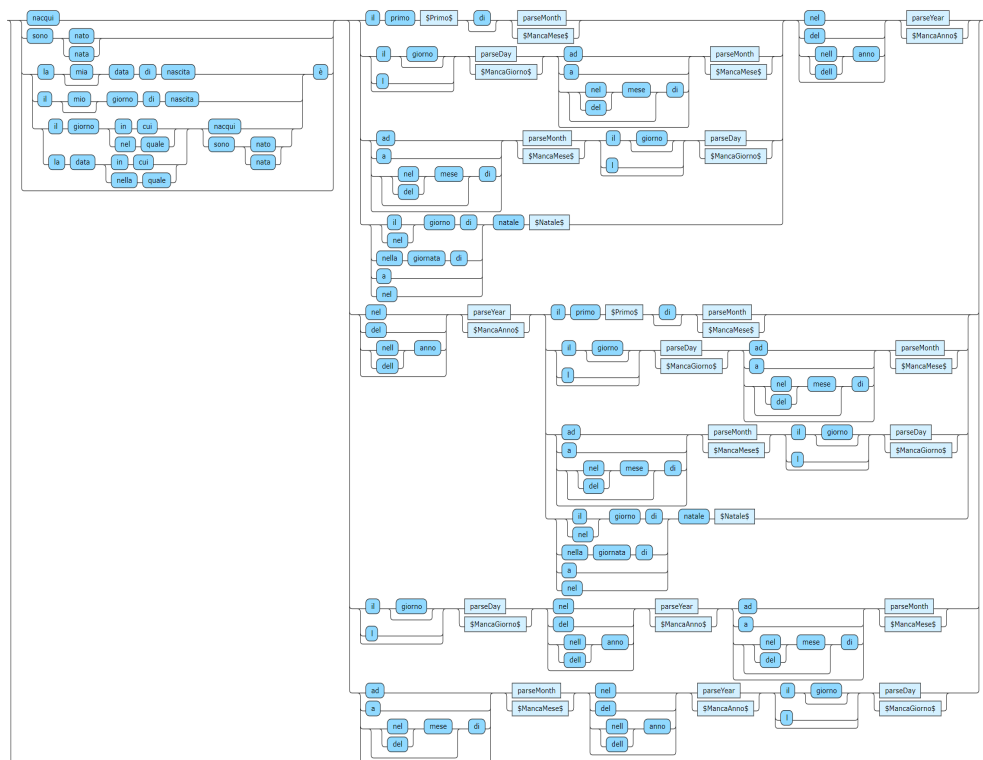
Le due porzioni di *grammatica* illustrate permettono globalmente di interpretare la data di nascita. Successivamente, seguendo lo stesso principio di separazione tra i frammenti di frase introduttivi e quelli di contesto, sono presentate le due porzioni di *grammatica* che illustrano la data di compleanno, differente dalla precedente per l'assenza dell'anno.

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

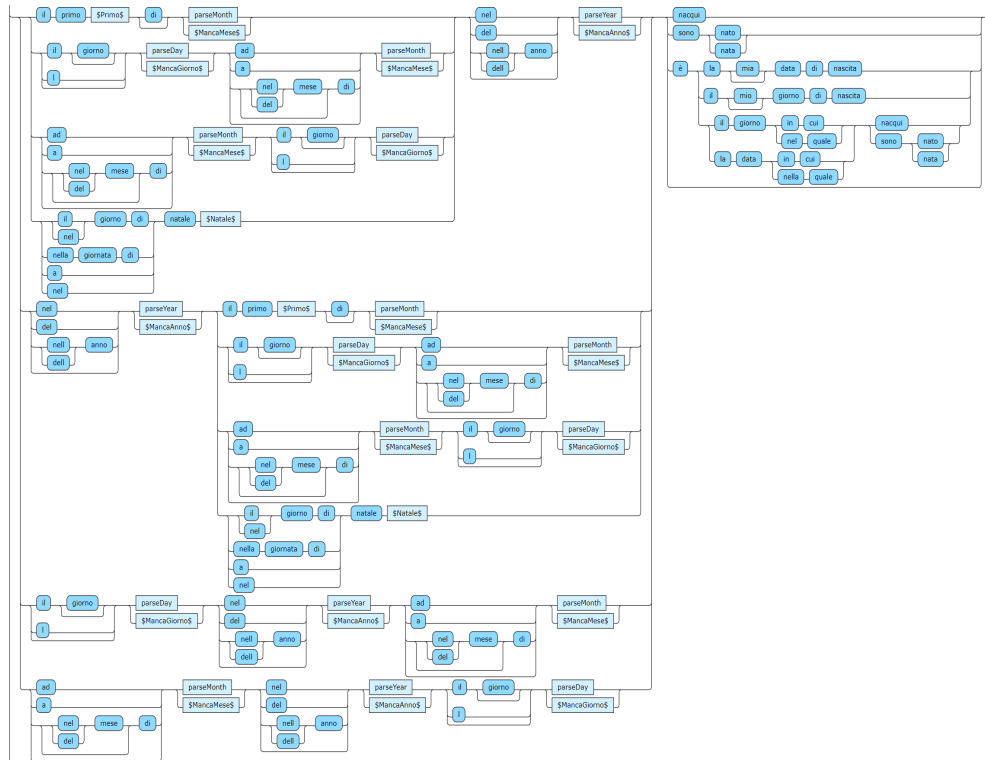
La prossima figura, invece, riporta la porzione che presenta prima la parte di contenuto e dopo quella introduttiva.

### 4.3.2 Capacità conversazionale

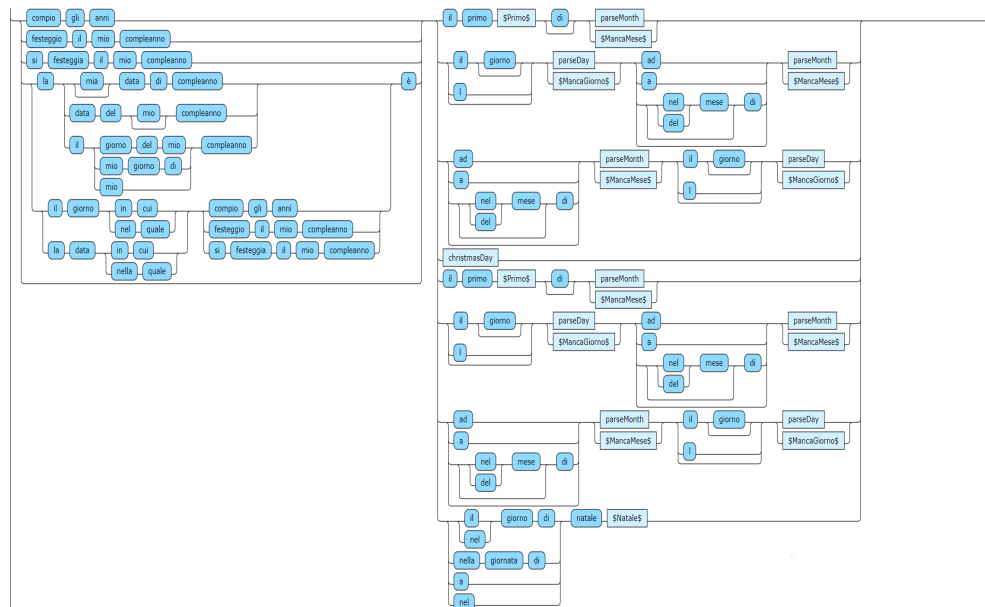
Per l'implementazione della capacità conversazionale il concetto fondamentale è la memoria. Infatti, mentre la *NLU* permette l'interpretazione del linguaggio naturale, la capacità conversazionale consente di mantenere il contesto durante l'intero dialogo. Ho deciso di tenere traccia dei dati che lo 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 gli eventuali dati



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

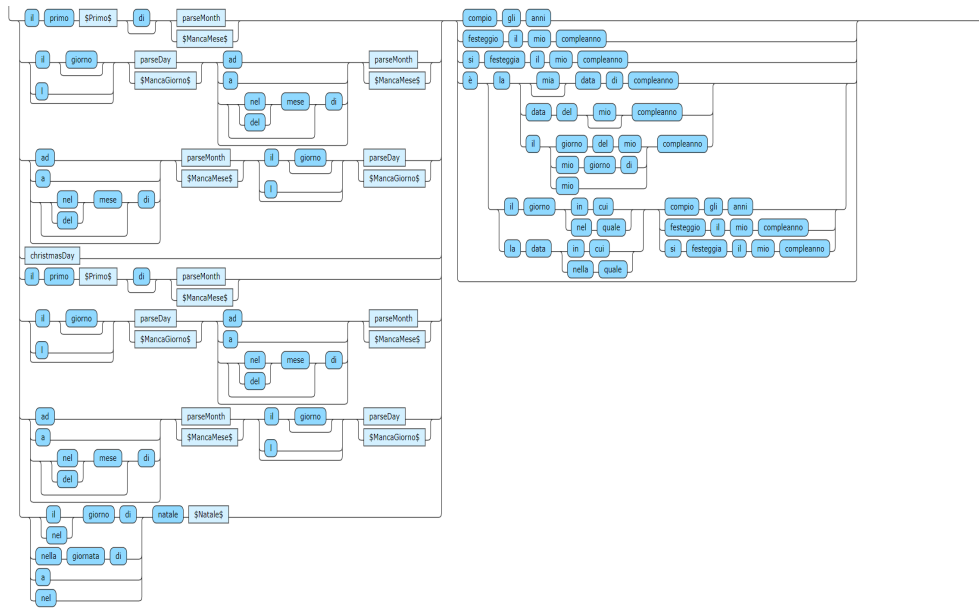


**Figura 4.4:** Diagramma railroad della grammatica per la data di nascita seconda parte



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





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

mancanti ovvero giorno, mese e anno.

La *grammatica* descritta in precedenza è stata progettata anche per riconoscere singole parti di contenuto e di conseguenza viene riutilizzata anche per l'implementazione della conversazione.

### 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 all'interno ed è illustrata nella seguente immagine.



**Figura 4.7:** Interfaccia grafica dell'applicazione

I componenti infatti 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 infine una casella di testo non editabile che permetta di visualizzare il comando che è stato riconosciuto.

### **Interfaccia vocale**

L'interfaccia vocale è stata progettata con l'obiettivo di garantire un'esperienza d'uso migliore possibile agli utenti. Ho utilizzato le nozioni apprese dalla documentazione degli assistenti virtuali analizzati per generare un'interfaccia vocale e sono:

\*

Tutti i componenti espressi nell'analisi sono quindi stati progettati. In particolare i comandi corrispondenti all'interruzione forzata dell'applicazione e alla richiesta d'aiuto che sono stati utilizzati risultano un'elaborazione di un sottoinsieme di quelli ricavati da un'analisi comportamentale di Zucchetti sugli utenti, qualora siano tranquilli e rilassati ma anche arrabbiati e agitati.

Per gli input dell'utente l'interfaccia vocale è realizzata automaticamente a partire dalla progettazione della *grammatica* per la quale sono comunque stati applicati i principi descritti mentre per l'output è stata personalizzata sulla base dell'elaborazione. Si è quindi deciso di fornire un set di frasi con significato uguale ma con una sintassi leggermente diversa da cui viene scelta al momento la frase di input secondo un algoritmo pseudo-casuale.

## **4.4 Codifica**

## **4.5 Test**

## **4.6 Sviluppi futuri**

Sarebbe stato possibile tenere traccia dei dati forniti dall'utente in modo permanente ad esempio all'interno di un database così che ad un nuovo utilizzo dell'utente l'applicazione si ricordasse di ciò che era stato detto precedentemente. Questo però pone alcuni vincoli quali un sistema di autenticazione dell'utente perciò diventa complesso.

# Glossario

**API** in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 21

**build** Nello sviluppo del software, *build* indica il processo di trasformazione del codice sorgente in un artefatto eseguibile.. 13, 22, 28, 41

**ERP** in informatica con il termine *ERP, Enterprise Resource Planning* (ing. pianificazione delle risorse d'impresa) è un software di gestione che integra tutti i processi di business rilevanti di un'azienda e tutte 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.. 1

**Firestore** *Firestore* è una piattaforma di sviluppo per applicazioni Web e mobile sviluppata da Firebase Inc. nel 2011 e acquisita da Google nel 2014. 14, 41, 42

**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 potenzialmente infinito di sequenze finite di simboli appartenenti ad un alfabeto finito.. 33–36, 39–41

**HyperText Transfer Protocol** In informatica *HTTP, HyperText Transfer Protocol* (ing. protocollo di trasferimento di un ipertesto) è un protocollo a livello applicativo usato come principale sistema per la trasmissione d'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.. 12, 14, 17, 31, 41

**IBM** L'*IBM, International Business Machines Corporation* è un'azienda statunitense, la più antica e tra le maggiori al mondo nel settore informatico. 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 ramo dell'elaborazione del linguaggio naturale nell'intelligenza artificiale.. 2–4, 7, 12, 14, 15, 17, 22–24, 28, 29, 35, 36

**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 che possono essere tradotti in bytecode sono Java, Kotlin e Scala.. 8, 42

**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 autorizzazioni ad un client previa approvazione dell'utente proprietario della risorsa cui si intende accedere. Rispetto alla sua versione precedente presenta una chiara divisione dei ruoli, implementando un mediatore tra client e server.. 31

**parser** In informatica il *parser* è un software che realizza il parsing ovvero un processo che implementa l'analisi sintattica. Più in dettaglio analizza un flusso continuo di dati in input e determina la correttezza della sua struttura grazie ad una data grammatica formale.. 34, 42

**PoC** Con *PoC*, *Proof of Concept* (ing. prova di fattibilità) si intende una realizzazione incompleta o abbozzata di un determinato progetto o metodo, allo scopo di provarne la fattibilità o dimostrare la fondatezza di alcuni principi o concetti costituenti. Un esempio tipico è quello di un prototipo.. 3–9, 31, 35, 42

**railroad** I *diagrammi railroad*, detti anche *diagrammi di sintassi* consistono in una rappresentazione grafica per grammatiche libere da contesto.. 34, 36, 42

**Software Development Kit** Un *SDK*, *Software Development Kit* (ing. pacchetto di sviluppo software) in informatica, indica un insieme di strumenti per lo sviluppo e la documentazione di software.. 12, 14, 15, 24, 25, 30, 32, 42

**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.. 9, 42

# Acronimi

**API** Application Program Interface. 41

**CSS** Cascading Style Sheets. 8, 9, 43

**ERP** Enterprise Resource Planning. 41

**ES6** ECMAScript6. 8, 9, 43

**HTML** HyperText Markup Language. 8, 9, 43

**IBM** International Business Machines Corporation. 41

**NLU** Natural Language Understanding. 42

**PHP** Hypertext Preprocessor. 9, 43

**RAM** Random Access Memory. 11, 43



# 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'intelligenza dei 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](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/>.