A Visual Framework for the End-User Development of Conversational Agents for Data Exploration

Politecnico di Milano, Scuola del Design, Corso di Laurea Magistrale in Design della Comunicazione

Relatrice	Maristella Matera
Laureanda	Ludovica Piro
Matricola	916299
A. A.	2019/2020

POLITECNICO
MILANO 1863

A Visual Framework for the End-User Development of Conversational Agents for Data Exploration

Politecnico di Milano, Scuola del Design, Corso di Laurea Magistrale in Design della Comunicazione

Relatrice Maristella Matera
Laureanda Ludovica Piro
Matricola 916299
A. A. 2019/2020

POLITECNICO
MILANO 1863

Ringraziamenti

I miei più sentiti ringraziamenti vanno alla professoressa Matera per la disponibilità e il tempo dedicatomi e per i sempre utili consigli che mi ha dato. Ringrazio anche il professor Desolda e Giulia Cosentino per avermi seguito nel corso della tesi.
Inoltre, ringrazio Emanuele Pucci e tutti i programmatori di Awhy per la disponibilità e la loro fondamentale partecipazione per la scrittura di questa tesi. Ringrazio anche Sara Mosca per aver sviluppato il front-end dell'interfaccia.

In ultimo vorrei ringraziare l'Ospizio di Brigida. Senza il loro supporto emotivo questa tesi non avrebbe mai visto la luce.

Table of Contents

List of Figures

List of Tables

Listings

Sommario

I chatbot per l'esplorazione dei dati costituiscono una classe di chatbot che, a partire dalle richieste formulate dall'utente in linguaggio naturale, permette di estrarre specifici dati da una base di dati strutturata. Questa tesi si è concentrata su CHATIDEA, un framework software che permette di generare chatbot per l'esplorazione dei dati in maniera automatica, ma richiede che il designer del chatbot conosca il formato di scambio dati JSON. Questo formato, infatti, è usato per definire alcune annotazioni alla base di dati, ossia i file che contengono le indicazioni necessarie per lo sviluppo delle conversazioni tra utenti e chatbot.

Questa tesi si presenta come un'espansione di CHATIDEA e ha come intento lo sviluppo di un'interfaccia grafica che permetta al designer del chatbot di svolgere la fase di annotazione dello schema della base di dati, senza dover compilare direttamente i file JSON. Per permettere agli utenti di modificare e creare nuove relazioni tra le tabelle di una base di dati, è stato disegnato un editor basato su una notazione a grafo. Invece, la stesura dei file di annotazioni avviene in un secondo editor che accetta input testuali dall'utente e fornisce un'anteprima di come risulterebbe la conversazione tra utente e chatbot. Le soluzioni presentate in questa tesi sono volte a rendere la fase di annotazione della base di dati accessibile anche a utenti che non conoscono il formato JSON e non conoscono approfonditamente il framework.

Abstract

Chatbots for data exploration are a class of chatbots that enables the end-user to explore a structured database and extract data from it through natural language queries. This thesis focuses on CHATIDEA, a software framework that allows to develop automatically chatbot for data exploration. However, it requires a manual intervention from a chatbot designer who has to know the JSON format, in order to define a set of JSON files called Database Schema Annotation. These files contain the necessary information to match the content of the database and the end-user requests made in natural language.

This work expands the CHATIDEA framework and has as its goal the presentation of a graphical user interface that covers the Database Schema Annotation procedure, which, right now, requires that the designer writes manually the JSONs. With the interface here presented, the designer will be able to generate the annotations without having to write directly the JSON files. The interface will allow the user to edit the relational database schema that has to be annotated through a graph-based editor. The actual annotation process is done through a form-like editor that accepts text inputs from the user. The user will also be able to see a preview of the conversation between chatbot and end-user resulting from the annotations. This work wants to make the annotation process easier and more accessible also to users that are not proficient with the framework and the JSON format.

1. Introduction

1.1. What Is a Chatbot?

We call chatbot or conversational user interface a computer program that operates as a user interface using primarily written or spoken natural language to communicate with a user (Janarthanam, 2017). The term chatbot comes from chatterbot a name coined by Michael Mauldin in 1994 (Mauldin, 1994) to describe this kind of program that is able to chat with a user via a text input. The user interacts with the conversational agent through a channel. It can be a voice channel, as in the case of personal assistants like Siri or Alexa, or it can be a textual channel, such as the various instant messaging app available right now as Telegram, WhatsApp, WeChat, LINE, Slack and so on. If the interaction is enabled by a voice channel, then the system architecture will contain also a speech recognition and synthetizer module to enable its transcription into text.

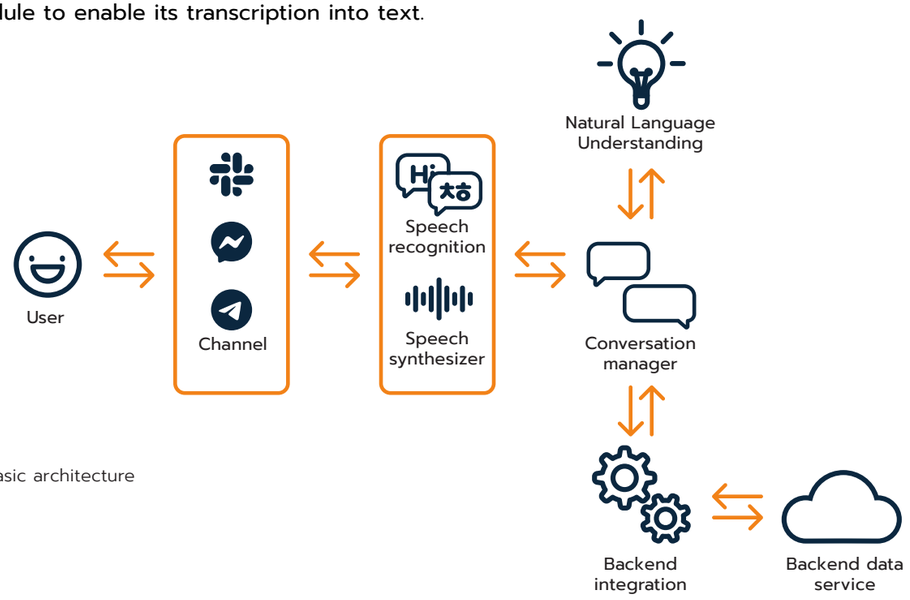


fig. 1
A diagram of the basic architecture of a chatbot
(Janarthanam, 2017)

In fig. 1 we can see a basic architecture of a conversational interface. The core of a conversational agent is the conversation manager: it is the module that takes the user input and controls how the system should respond. It also maintains the conversational context in order to make the chatbot able to

carry out a conversation for several turns. In chatbot that can understand natural language, the conversation manager interacts also with a Natural Language Understanding module, NLU, that is responsible of the translation of the user utterances into a semantic representation, consisting of intents (what the user wants) and entities (the relevant nouns in a sentence). Lastly the Conversation Manager interacts also with backend modules that could be a database or an online data source that can be queried to answer a user question or carryout a task like making a reservation.

The idea of a computer program that could behave like a human dates back to the '50s, when Alan Turing proposed the *Turing test* to determine if a machine can be mistaken for a human by another human (Turing, 1950). The program ELIZA (Weizenbaum, 1966) is one of the first examples of such a program: developed in 1966 by Weizenbaum, ELIZA was able to pick up keywords in input text and use them in a vague question given back as a response without a real understanding of what the user was saying. Eliza was also the first program to win the Loebner Prize, a prize founded in 1991 by Hugh Loebner to award AI programs able to be human-like and beat the Turing test. After ELIZA other notable programs developed to beat the Turing Test and mimic humans were PARRY (1972), JabberWacky (1988), A.L.I.C.E (1995), and Mitsuku (2005).

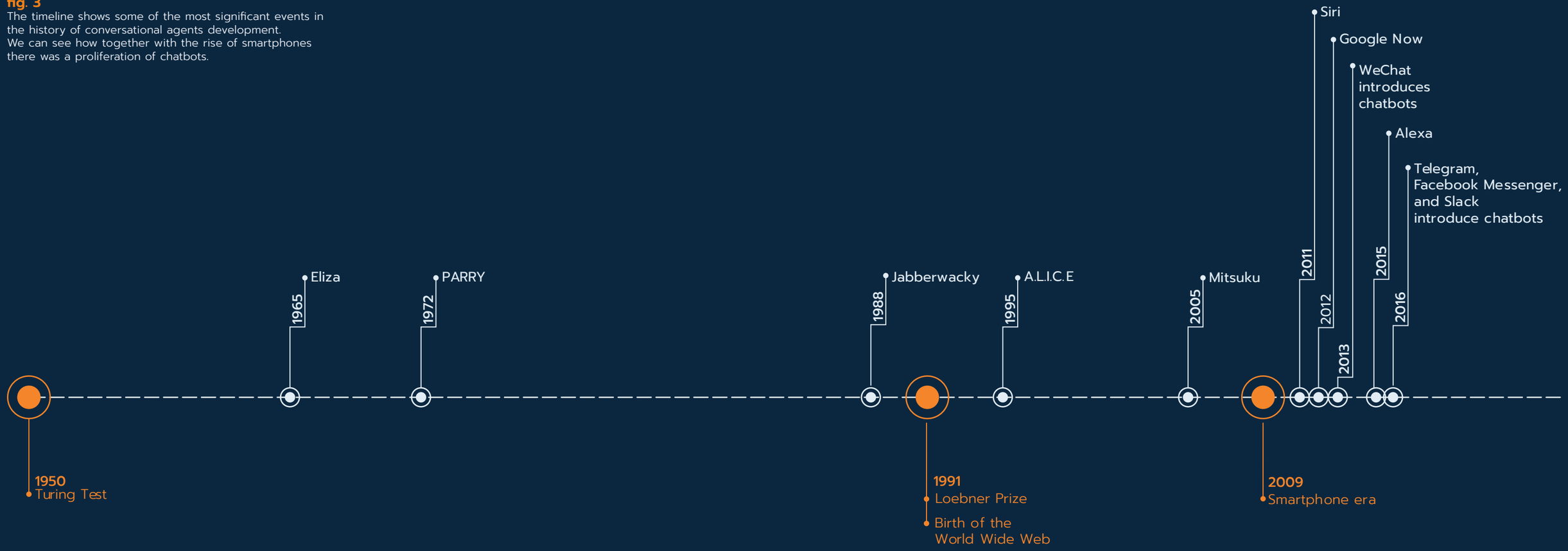
fig. 2
Watson taking part in an episode of Jeopardy
(Carol Kaelson/Jeopardy Productions Inc., via Associated Press)

With the advances in artificial intelligence, natural language processing and speech recognition, conversational agents have become more "intelligent", being able to complete tasks and better understand conversations. In 2010 IBM developed Watson (Ferrucci et al., 2010), a question answering AI built on data from encyclopaedias, dictionaries, and other structured and unstructured data sources like Wikipedia. Watson was designed to participate in a game show called Jeopardy which it won.

At the same time, there was the rise of a new type of chatbot: virtual assistants, bots that support speech recognition and interact with the user through spoken natural language. These assistants can do small talk, answer user's questions, and interact with other applications on the device or outside of it to, for example, set appointments or tell the time. The first to be considered the first voiced-enabled virtual assistant was Apple's Siri in 2011. Siri was then followed by other popular assistants, like Microsoft's Cortana, Google Assistant by Google, and Amazon's Alexa.

As of late, the ever-growing popularity of instant messaging applications has given text based chatbot a newfound popularity, so much that in an article on the technology online magazine Chatbot Magazine, it is reported that in China, some brands create a chatbot on WeChat even before launching a website (Jerry @Rocketbots.io, 2017).
In fact, starting from 2013, WeChat, then followed by Telegram, Slack and Facebook Messenger in 2016 has started to support bots and have also opened their APIs to third party developers, leading to a proliferation of text-based chatbots (Klopfenstein et al., 2017).

fig. 3
The timeline shows some of the most significant events in the history of conversational agents development.
We can see how together with the rise of smartphones there was a proliferation of chatbots.



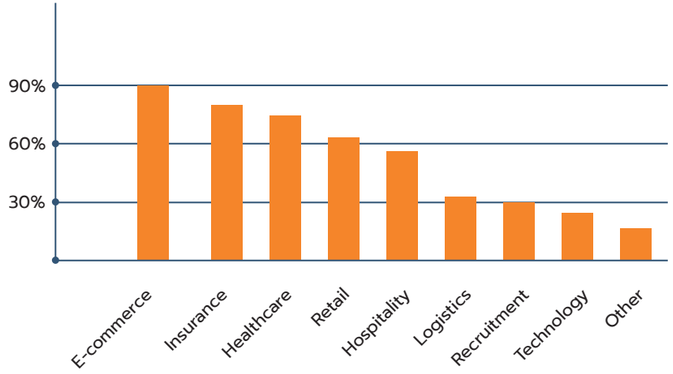
The bots, created by end-users and brands alike, are used for a variety of ends, from entertainment to providing services like telling the weather, booking rooms, or retrieving information. Furthermore, conversational agents are now starting to be used for automatization of customer care, CRM[1], and marketing. On retail websites, they are also used as a welcome mat, providing a fast way to answer simple questions from first time user, while collecting customer information at the same time. Some of the other sectors where they are mostly employed include, banking, legal and third sector, and the health sector (Janarthanam, 2017).

fig. 4

Industries that will benefit the most from chatbots

According to a survey conducted by Mindbowser among more than 300 individuals from different industries

(Mindbowser, 2017)



Together with the spread of chatbots, there has been also a significant growth in terms of tools and frameworks available for the development of conversational interfaces. This new ecosystem of tools targets both experienced developers and non-programmers that want to build a chatbot for private or business uses and is, therefore, very diverse regarding the user interfaces. But, despite the diversity in terms of interface design, most of the tools share the same chatbot building process: intent matching and entity extraction.

At a high level, if we consider tools for non-programmers, the way that the chatbots are built is by defining a set of intents and entities. Intents are phrases that represent what the end-user wants from the chatbot, their intention with the message like buying a product or asking what the time is. To handle a complete conversation the chatbot designer defines multiple intents each for a specific end-user need.

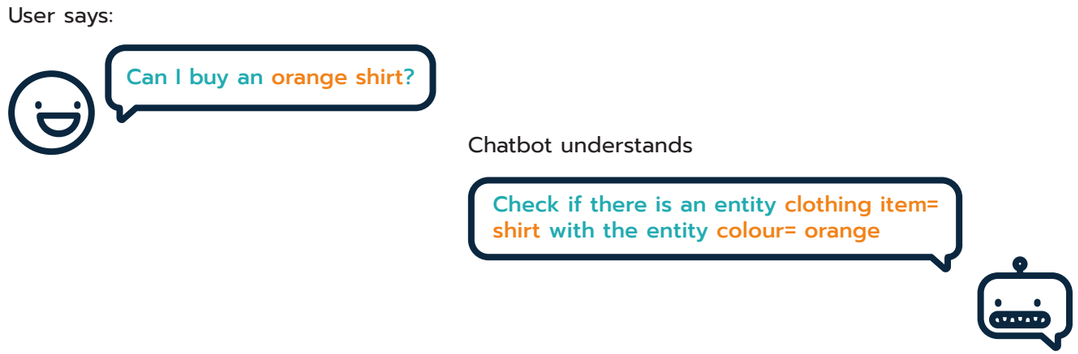
Usually, the way that each *intent* is defined is by listing several example phrases, often referred as *training phrases*, that are variation of the same sentence and thus intention of the user.

1 Customer Relationship Management

Together with this listing, the designer also defines the response that the chatbot should give in return.

Entities, instead, are critical pieces of information that the chatbot must be able to recognize in a sentence. *Entities* are defined like variables: they have a name and a list of possible values. An example of an entity could be the list of the colours available for a shirt or the kind of clothing items available, such as shirts, trousers, jumpers and so on. So, for example, if we are building a chatbot to buy clothes, one user utterance could be "Can I buy an orange shirt?". In this utterance the *intent* is the action "buy", whereas the *entities* are "orange" and "shirt."

fig. 5
Example of intent matching and entity extraction



1.2. Conversational UIs and Data Exploration

What was here described is a process that leads to the generation of a chatbot oriented towards absolving specific tasks, like telling the weather or managing ticket sales. With this process the developer acts like a screenwriter, creating a sort of script of the conversation with the end user where all the questions and answers are written from scratch. But what if a developer wants to build a chatbot not to assist the end-user in specific tasks, but to navigate a database? A chatbot that guides the user through the database and can retrieve information from it, without knowing how to write queries in a technical language such as, for instance, SQL.
The developer would need a way to match the data contained in the database to the end user utterances. Not only that, but the chatbot would need to be able to translate the end user requests into actual queries to the database and then, answer with the correct data.

Even before chatbots, there was a great interest in the development of conversational interfaces to access large databases. Natural language, in fact, provides a simple and accessible way to query database, enabling users that do not have any formal query language knowledge to perform queries and extract information from a structured database (Androutsopoulos et al., 1995). Natural language interfaces have been developed to query databases since the 70s. An early example is LUNAR, which was developed in 1972 (Woods et al., 1972). LUNAR was a prototype of a natural language interface to a database (NLIDB) containing chemical analyses of moon rocks. The early applications, such as LUNAR, were tailored on the database they were developed for, highlighting one of the main issues with such applications: portability. Developing an interface able to query any database is, in fact, challenging, for whenever the interface has to be used to query a new database, a reconfiguration is needed in order to teach the system the new words and concepts. Nonetheless, generic domain applications not tailored on a specific database have been developed.


```
SENTENCE:
(WHAT IS THE AVERAGE COMPOSITION OF OLIVINE)
PTIMING:
1168 CONSES
4.9 SECONDS
PARSINGS:
S Q
  NP DET THE
     N AVERAGE
     NU SG
     PP PREP OF
        NP DET NIL
           N COMPOSITION
           NU PL
           PP PREP OF
              NP DET NIL
                 N OLIVINE
                 NU SG
  AUX TNS PRESENT
  VP V BE
     NP DET WHQ
        N THING
        NU SG/PL

ITIMING:
2285 CONSES
9.877 SECONDS
INTERPRETATIONS:
(FOR EVERY X8 / (SEQ MAJORELTS) : T ; (FOR THE X4 / (SEQL (AVERAGE
X5 / (SSUNION X6 / (SEQ SAMPLES) : T ; (DATALINE (WHQFILE X6) X6 (NPR*
X7 / (QUOTE OLIV)) X8)) : T)) : T ; (PRINTOUT X4)))

SIO2  (36.93518 . PCT)
TIO2  (.1544509 . PCT)
AL2O3  (.1236187 . PCT)
FE2O3  (0.0)
FEO  (28.97409 . PCT)
MNO  (.3488543 . PCT)
MGO  (33.82487 . PCT)
CAO  (.3093421 . PCT)
K2O  (0.0 . PCT)
NA2O  (.1381333 . PCT)
↑L
```

fig. 6
An example of interaction with LUNAR. The second line in the script is the user query; the middle sector shows the parsing of the query and the interpretation made by the system; while the answer is at the bottom of the page. (Woods et al., 1972)

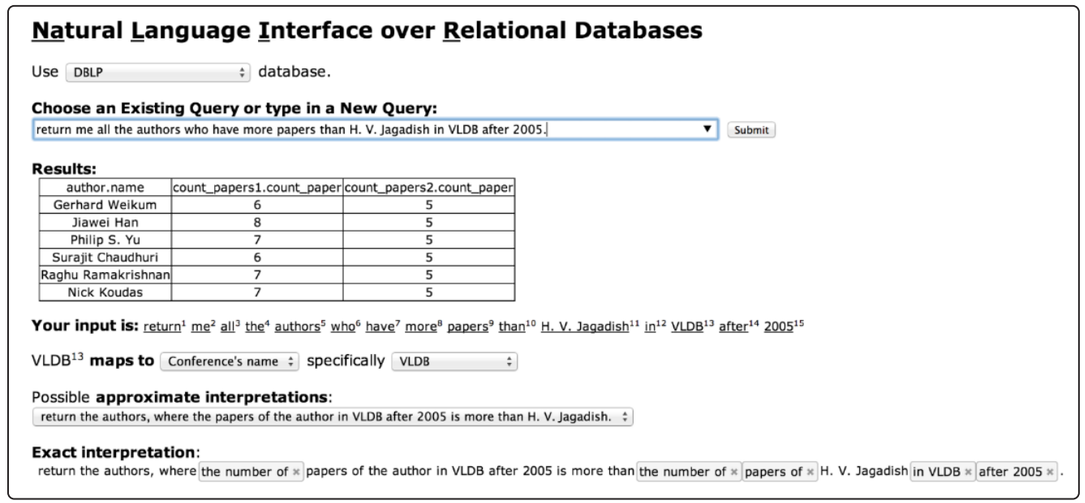


fig. 7
A window of NaLIR (Li & Jagadish, 2014).

The most recent example of such application being NaLIR (Li & Jagadish, 2014). NaLIR is presented as a *generic interactive natural language interface for querying relational databases.* The tool accepts complex sentences from the user and builds the final query with an interactive approach. The user is called to confirm if the query was understood correctly until the correct and final query is generated and a response is given.

Actual chatbots to assist in database exploration have been developed in more recent times, like Intellibot, a dialogue-based chatbot for the insurance industry (Nuruzzaman & Hussain, 2020). Intellibot is a domain specific chatbot created for customer service developed and trained on three different databases: the Cornell movie database to get training material for conversing using the English language; one custom database about the most common words and expressions used in an insurance company; lastly, it was trained on a credit-card insurance company database. The extensive training makes it able to respond precisely and engage in a well-structured dialogue with the user. Based on what was asked, Intellibot can choose a different strategy to form a response. If the question is one of the pre-set ones, the chatbot will simply respond with the predefined answer. In more complex scenarios, the chatbot is able to generate a response by searching the database or the Internet.

However, Intellibot is still a custom made chatbot, developed for a domain-specific conversation. If we look at the frameworks commercially available right now to develop chatbots,

it seems to still be missing a common and general approach for the development of such data-driven chatbots. In fact, while some tools available at the moment may support retrieval of information from spreadsheets (Canh, 2018), or from the Internet thanks to webhooks, they do not contemplate a way to develop a chatbot directly starting from the content of a database and using it as the source material for the generation of intents and entities.

1.3. Contribution

Right now in the industry there seems to be no End-User Development tool to generate chatbots starting from a structured database. This works aims at filling this gap by presenting a graphical user interface for the end-End-User Development of such chatbots. To do so it capitalises on CHATIDEA, a framework for the rapid prototyping of chatbots for data exploration, that was developed in previous works. CHATIDEA currently lacks a visual tool and, thus, the chatbot must be coded by a programmer. This work proposed a new GUI for CHATIDEA based on a conversational paradigm. The tool here proposed will allow non-programmer users to develop a chatbot based on a database by completing conversation patterns that mimic the interaction between users and chatbot.

1.4. Thesis Structure

In the following chapters we will see, firstly, what the state of the art is for chatbot development platforms, as in what are the main development frameworks available, how they work and, most importantly, what model of interaction with the user they employ for their user interface. Together with that, the concepts of End-User Development and a brief taxonomy of chatbots will be given, describing the different types of chatbots that can be developed as of now.

After that, the third chapter will present the CHATIDEA framework explaining its general architecture and the main abstractions, the Database Schema Annotations, that are at the basis of this work.

Then the fourth chapter will present the approach taken for the development of the user interface, from the chosen interaction paradigm to the main challenges that were encountered during the design on the interface.

The fifth chapter, instead, will describe the visual identity of the UI, form the choice of colours, to typography, and layout of the different sections and graphic elements.

The sixth chapter will describe the information architecture of the interface, going into the details of how each window is structured and how each task can be performed by the users.

The seventh chapter will cover the user studies undertaken to test the usability of the tool and to validate the approach. The last chapter of this thesis will cover what could be covered in future work.

Lastly, the eighth chapter is the conclusion of this thesis and contains a synthesis of the work and the possible future improvements that can be made in following iterations of the interface.

1.5. Glossary

1.5.1. Conversational Agents concepts

Conversational agent: also referred to as chatbot, it is a computer program that operates as a User Interface using primarily written or spoken natural language to communicate with a user.

NLU: It stands for Natural-language understanding and is a component of natural language processing, NLP, which enables computers to understand and interpret text written in human language.

Intent: it represents the purpose of the user's interaction with a chat-bot; for instance, from the phrase "What time is it?" the intent, if defined, could be to know the current time.

Entity: it is a word or a set of words that represent a concept in a given utterance; for instance, in the phrase "I live in Milan" the word "Milan" may be an entity related to the concept of location.

1.5.2. Relational Databases concepts

Query: a query is a request for data from a database. The request can be made to retrieve data or manipulate it.

Relationship: in a relational database a relationship is an association between two tables.

Attributes: in a relational database attributes are essentially the columns of the table. They represent the characteristics or properties of an object in the table.

Primary key: it is an attribute, a column, that can uniquely identify a row in a table. Often, a primary key is a numerical id.

Foreign key: it is an attribute in a table that refers to the primary key belonging to another table. With the primary keys they are attributes that act as links between two different tables.

2. State of the Art

2.1. Summary

In this chapter, we will start by giving a brief categorization of the different types of chatbot existing right now, focusing specifically on the difference between task-oriented chatbots and non-task-oriented ones. Then, to better frame the chatbot development platform, a brief explanation of what is End-User Development will be given. Lastly, we will analyse some of the current available frameworks for the development of conversational agents, with a particular focus on the graphical user interfaces of their platforms for the building of the conversation architecture. The analysis will focus on finding the most common and relevant practices and on highlighting the possible interesting features for this research.

2.2. Chatbot Taxonomy

To understand better the chatbot development platforms, first, it is important to give an overview of the different types of conversational agents. Chatbots, in fact, are a very diverse category of programs and can be classified according to different criteria: their interaction mode, their design approach, knowledge domain or goal (Hussain et al., 2019) designed to interact with users using natural language or text in a way that the user thinks he is having dialogue with a human. Most of the chatbots utilise the algorithms of artificial intelligence (AI).

On a high level, generally, chatbots are mainly classified based on goals. Thus, we talk about **task-oriented chatbots** and **non-task oriented or conversational chatbots** (Chen et al., 2017). The former are programs that mainly serve in helping users to complete a specific task in a specific domain. They use rules and NLP, but generally the interaction with the users and are highly specific. Task-oriented chatbots are most commonly used in support and service functions, as they can handle common questions and be used in reservation-making scenarios or as interactive FAQs, although more complex examples exist, like

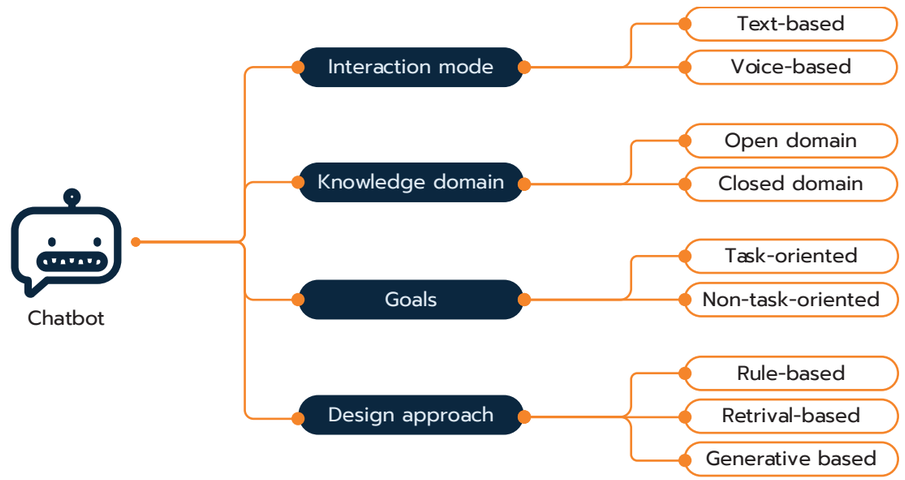


fig. 8
Diagram of chatbot classification. (Hussain et al., 2019)

Apple's Siri or Amazon's Alexa. Non-task oriented or conversational chatbots, instead, are more sophisticated and interactive. Often referred to as digital assistants, they are designed for extended conversation, not limited to practical purposes, but able to also handle chit-chat and entertainment. These chatbots are contextually aware can learn from the interactions with users thanks to machine learning. Pandorabots' Mitsuku[2] and Replika[3] are examples of such conversational agents.

Another important distinction to describe is the design approach: chatbots can be **rule-based, retrieval-based, generative-based** (Peng & Ma, 2019). The first method is most used for task-oriented chatbots, while retrieval and generative methods are used for non-task-oriented agents. Both retrieval and generative-based chatbots are **corpus-based**, meaning that they mine human-human conversations. In fact, they are data-intensive systems, that are trained using extensive datasets from various sources. These datasets include transcripts of telephone conversations, movie dialogue (Danescu-Niculescu-Mizil & Lee, 2011), and, more recently, content from social media like Twitter (Ritter et al., 2010), Reddit (Roller et al., 2020) or Weibo.

Rule-based chatbot, also referred to as decision-tree bots are built on manually constructed rules. The rules can be based on an "if-then" logic or involve pattern matching. The bot contains a knowledge base with documents, where each document contains an input-pattern and an output-template. When the user gives an input, if that matches a pattern stored in the knowledge base, the chatbot will send as a response the corresponding output-template. The pattern could either be a phrase such as "What's your name?". Usually, these pattern and template pairs are hand-crafted (Cahn, 2017). Often, they leverage keywords that, when recognized, trigger a specific response. Chatbots developed for services, such as booking flights or making

2 https://chat.kuki.ai/
3 https://replika.ai/

reservations are often rule-based. Platforms such as IBM Watson and Dialogueflow, that we will see in detail later in the chapter, offer the tools necessary to build such rule-based chatbots (Peng & Ma, 2019). The complexity of the interaction can also vary within the group. Some chatbots may offer exclusively a multiple-choice style of interaction, where the user can only choose between the available buttons to interact with the agent. Other, may support a more conversational approach and accept text input. Despite this, their understanding is still limited to the task they were designed for, as they cannot go beyond the rules they were designed by.

Retrieval-based chatbots rely on a conversational repository to select the response that best matches the user query. This means that the chatbot does not generate new text, but only reuses responses from a fixed corpus of information. With respect to rule based chatbot, retrieval- based one need less hand-crafted features. However, even though they are usually able to provide correct and diverse responses, they are also prone to give inappropriate responses if they fail to correctly match the words in the user input. Nonetheless, retrieval methods are the most reliable and most commonly used in question-answering chatbots for specific domains, like travel or healthcare.

To answer user requests, generative-based chatbots synthesize a new sentence word by word from scratch. They are mainly used for chit-chat or open-domain chatbots, meaning chatbots with which the conversation is not constrained to a specific knowledge area like domain-specific chatbots. Generative-based chatbots are generated with data-driven methods that do not contemplate an explicit model of dialogue structure. Instead, the model learns to converse from the corpus of human-to-human conversational data. Furthermore, thanks to the Deep Learning techniques they are developed with, generative-based agents can generate answers while also being context-aware, meaning that they are able to remember and use information extracted from past interactions to generate the answer. Despite their human-like capabilities, they are still prone to give generic responses if not properly trained. Their performance, in fact, is highly dependent on data quality, quantity, and diversity of topic in the data, especially if they are meant to be employed for domain specific conversations. Nonetheless they are at the centre of contemporary research.

2.3. End-User Development

The platforms that will be described fall into the category of End-User Development tools, a field that encompasses different research topics from Human-Computer Interaction (HCI), software engineering, and artificial intelligence (AI). It aims at empowering people that usually are not skilled in programming, the end users, to develop and adapt the system they use themselves. As computers have progressively become more widespread, in fact, the user base has also increased both in numbers and diversity of case of use. Consequently, their software needs have become more complex and varied, leading to the need for software customization and, together with that, to the need of new ways and paradigms designed for users not formally trained in programming. Quoting Lieberman (Lieberman et al., 2006):

> *EUD can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact.*

The entity of the modification can vary. On one side, in fact, End-User Development, also referred to as EUD, enable users to do activities of parameterization or customization. These activities do not modify the source code of the software, but allow the users to choose alternative behaviours already available within the application they are using. This way users can tailor the interaction between them and the application to their needs. However, EUD is more properly referred to another set of activities that regard program creation and modification, meaning activities by which users can modify or create from scratch a software. Visual programming[4] environments, macros, and scripting languages are examples of these approaches.

The EUD tool most widely used is the spreadsheet. In fact, while one can be unaware of it, spreadsheets are a type of End-User Development tool, as the formulas are indeed programs. The formulas use as input "variables" cell names and the result is an output value. More specifically, spreadsheets are an example of End-User Programming (EUP), a branch of End-User Development. End-user programming is a term popularized by Nardi (Nardi, 1993) to describe the activity of:

> *Programming to achieve the result of a program primarily for personal, rather public use.*
> (Ko et al., 2011)

End-user programming tools are used by people with

4 Visual programming refers to a set of interaction techniques and visual notations for expressing programs. Visual programming tools encode the semantics of the program in graphical properties like colour, size, position and so on.

Class of people	Activities of programming and tools and languages used
System administrators	Write scripts to glue systems together, using text editors and scripting languages
Interaction designer	Prototype user interfaces with tools like Flash
Artists	Create interactive art with languages like Processing
Teachers	Teach science and math with spreadsheets
Accountants	Tabulate and summarize financial data with spreadsheets
Actuaries	Calculate and assess risks using financial simulation tools like MATLAB
Architects	Model and design structures using FormX and other 3D modelers
Webmasters	Manage databases and websites using Access, FrontPage, HTML, Javascript
Health care workers	Write specifications to generate medical report forms
Videogame players	Author "mods" for first person shooters, online multiplayer games, and The Sims

table 1
In the table we can see some of the most common uses of End-User Programming (Ko et al., 2011)

expertise in other fields, such as education or graphic design, working towards goals for which they need computational support. For instance, a teacher might write a grading spreadsheet to

speed up the grading process, or a motion designer might write a script to automate parts of an animation. The programs developed through EUP can be extension or completely new applications running independently from the already existing application. Other examples of End-User Programming tools are visual programming environments. In visual programming, the user, instead of a text editor, users a visual paradigm that expresses through the layout and the graphical elements the program's semantics. Semantics can be encoded in various attributes, such as colour, size, or position. Examples of such applications are Peridot (Myers, 1990), Scratch, or Alice. Peridot is an experimental tool for designers that allows users to create user interface components starting from a drawing and without conventional programming. Scratch (Resnick et al., 2009) is an environment aimed mainly at children that uses a building block visual metaphor to represent the programming syntax. Lastly, Alice (Conway et al., 2000) is a prototyping environment for 3D graphics programming. To be used, 3D programs at the time required advanced mathematical and programming knowledge. The tool. Alice instead, used Python to program behaviours (animations), and did not require particular mathematical knowledge to be used. The user could add to the workspace, called *opening scene*, from a library the 3D objects, and move them with the mouse. Then, they could program the behaviours they wanted to add, by using the *script* tab at the top of the

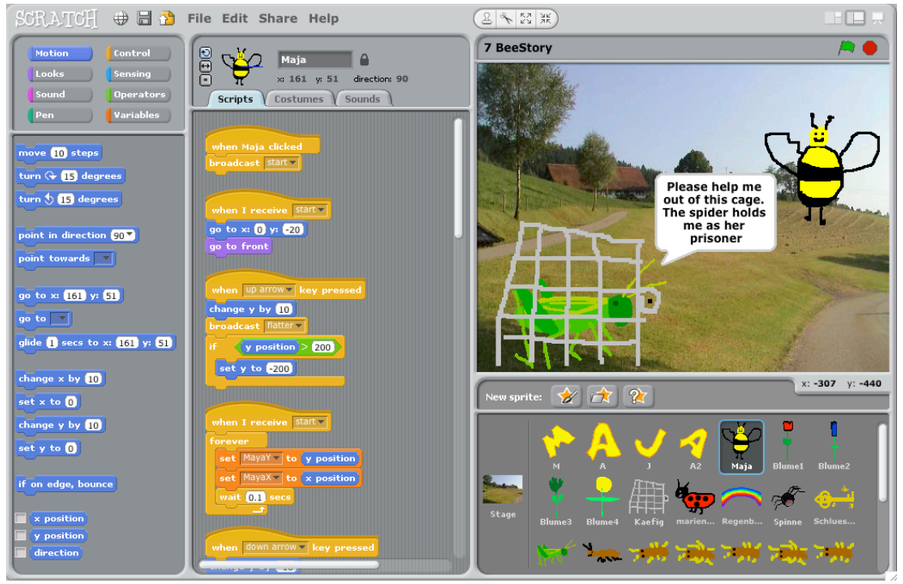


fig. 9
The GUI of Scratch (Resnick et al., 2009)

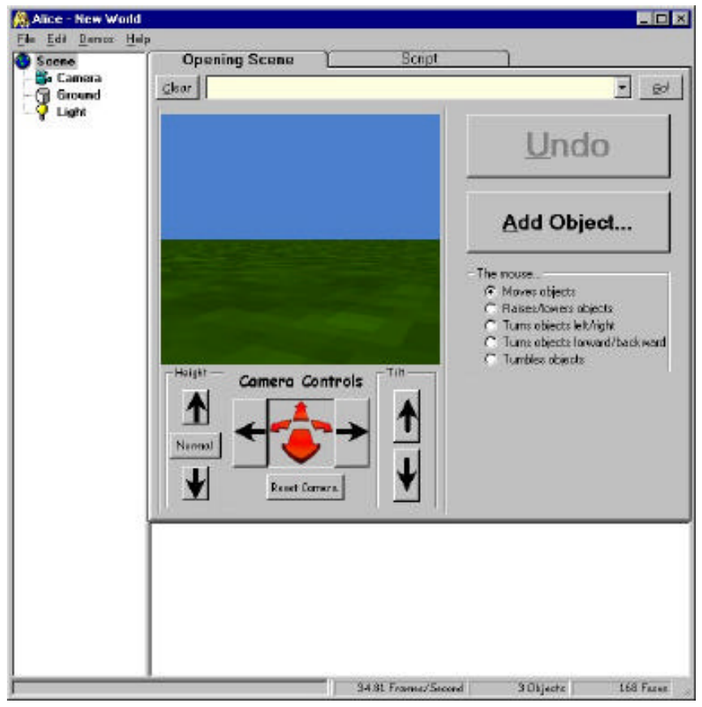


fig. 10
The GUI of Alice (Conway et al., 2000)

GUI, as we can see in fig.

Many chatbot development platforms offer analogous visual programming environment that use flow charts and other visual metaphors to support the user in the development process.

2.4. Chatbot Frameworks

Parallel with the rapid growth in popularity of conversational agents, many digital companies and start-ups started developing tools to program them. Right now, there are many platforms on the market, and they are catered towards different user segments and use cases. From business-oriented platforms, offering specific marketing and CRM features, to developers, to non-programmer individuals who want to build a chatbot for private or business-related needs. Although a formal and universally accepted classification of these platforms has yet to be made, many websites offer every year comparisons and rankings (Patil, 2020), (Brooks, 2018), (Rehan, 2018). From these rankings some key dimensions can be extrapolated to propose a classification: *context, implementation, intelligence, channel integration* (Monsters Data, 2019), (Wouters, n.d.).

2.4.1. Context

Context indicates whether the platform is *context-specific* or *general-purpose*. Context-specific platforms only offers templates for a selected range of business-related activities as conversational marketing, conversational support, financial services, e-commerce. Some examples of such platforms are Intercom, WotNot, ManyChat, and Mobile Monkey. All three of them offer marketing tools and can be integrated with other CRM tools. General-purpose platforms, instead, offer a more flexible environment that enables the user to freely create a chatbot independently from its scope. Such platforms may still offer templates for the most common use cases, like greetings, and marketing activities. Examples of such platforms are Google Dialogflow, Botpress, and IBM Watson assistant.

2.4.2. Implementation

Implementation is one of the main factors by which online blogs classify chatbot platforms. It refers to the method that the user uses to build the chatbot. The platforms exploit mainly three different methods: *code-based*, *no-code*, or a *hybrid* approach. *Code-based* platforms require the user to actually

code the chatbot. This also means that, while offering NLU and integrations with messaging channels like Slack or Telegram, they do not offer a GUI to design the conversation, like other platforms do, and that every step of the development is handled by writing code with a text editor. Platforms like RASA, Pandorabots and Botkit are examples of code-based platforms. *No-code*, on the contrary, is the approach typical of platforms that offer a visual programming tool that enables the user to develop the chatbot without the need to write any code. Usually, they exploit flowchart and tree diagrams to model the conversation structure. Some examples are FlowXO, Landbot, Flow.ai. It's important to note, that while these platforms may be advertised as no-code, they may still support custom code, although the chatbot can still be built without it. Lastly, some platforms offer a hybrid approach that combines a GUI with the flexibility of supporting code, which grant experienced user the possibility to add more features and create more complex agents. Examples of such platforms are the already mentioned Dialogflow and IBM Watson.

2.4.3. Intelligence

This dimension refers to the AI capabilities of the platform. Platform can be *rule-based,* or *AI powered*. No-code platforms tend to be rule-based. Although some do support NLP, like Flow.ai, they often lack Natural Language Processing support and offer the possibility to only create rule-based chatbots that recognize specific keywords. However, some platforms that do not support NLP, like FlowXO, can be integrated with external NLP services like Dialogflow. On the contrary, most code-based platforms are AI powered and natively support NLP or can be integrated with NLP services, like in the case of Botkit that can be integrated with RASA, Dialogflow, IBM Watson and many more. Lastly, also hybrid platforms are generally AI powered. Moreover, as some of them are products of tech giants like Google and IBM, they offer advanced NLP services.

2.4.4. Channel integration

This last dimension refers to what communication channel can the developed chatbot be deployed. The platforms can be *cross-channel* or *single-channel*. The majority of chatbot

Chatbot platform	Context	Implementation	Intelligence	Channel Integration
Amazon Lex	General purpose	Hybrid	AI powered	Cross-channel
Botkit	General purpose	Code-based	AI powered	Cross-channel
Botpress	General purpose	Hybrid	AI powered	Cross-channel
Botsify	Context-specific	No-code	Rule-based	Single-channel
Chatfuel	Context-specific	No-code	Rule-based	Single-channel
Dialogflow	General purpose	Hybrid	AI powered	Cross-channel
Flow.ai	Context-specific	No-code	AI powered	Cross-channel
FlowXO	Context-specific	No-code	Rule-based	Cross-channel
IBM Watson Assistant	General purpose	Hybrid	AI powered	Cross-channel
Landbot	Context-specific	No-code	Rule-based	Single-channel
ManyChat	Context-specific	No-code	Rule-based	Single-channel
Mobile Monkey	Context-specific	No-code	Rule-based	Cross-channel
Pandorabots	General purpose	Code-based	AI powered	Cross-channel
RASA	General purpose	Code-based	AI powered	Cross-channel
WotNot	Context-specific	No-code	Rule-based	Cross-channel

table 2
In the table we can see a systematization of the classification here proposed

platforms is cross-channel: the agents can be deployed on the most common instant messaging services, such as Telegram, WhatsApp, WeChat or Facebook Messenger. They can support also proprietary website integration or Slack. Single-channel platforms, on the contrary, support only one channel. Some platforms, such as Chatfuel or Botsify, or ManyChat support only Facebook Messenger, while Landbot is only for websites.

As this work is aimed at the design of a graphical user interface, for this next part of the chapter, code-based frameworks, such as RASA, Pandorabots, and Botkit, will not be further discussed, as they do not provide a GUI for the chatbot development. Hybrid tools, instead, will be taken into account despite being targeted to a more experienced audience than the one of tool prented in this work, because they are the most commonly used AI powered tools in the industry and offer interaction paradigms different from no-code platforms.

2.4.5. Dialogflow

Implementation: hybrid
Context: general purpose
Intelligence: AI powered
Channel integration: cross-channel

Dialogflow[5] is an end-to-end tool powered by Natural Language Understanding offered by Google to design conversational interfaces, both text chatbots and voicebots. It offers NLU/NLP, and it is supported by Google's machine learning infrastructure. Dialogflow's conversational agents can be connected to other apps or digital communication channels such as Slack, Telegram, Facebook Messenger, Kik, LINE. The voicebots can be used with Google Assistant, Amazon Alexa, and Microsoft Cortana. Dialogflow provides template for small talk chatbot and some common scenarios. Chatbot built from scratch can be rule-based or use Machine Learning for intent matching. *Intents* and *entities* are the two fundamental concepts in the system.

Intents
An intent is set by giving it a name and then providing a set of examples of what the user could say, that is called *training phrases*. In addition to that, the designer must provide a set of possible *responses* that the chatbot could give as an answer to the user. Lastly, to make it so that the chatbot can recognize some nouns or other elements of the *training phrases* as data, said elements can be tagged as a parameter. When tagged as a parameter the annotated element is matched to an *entity*.

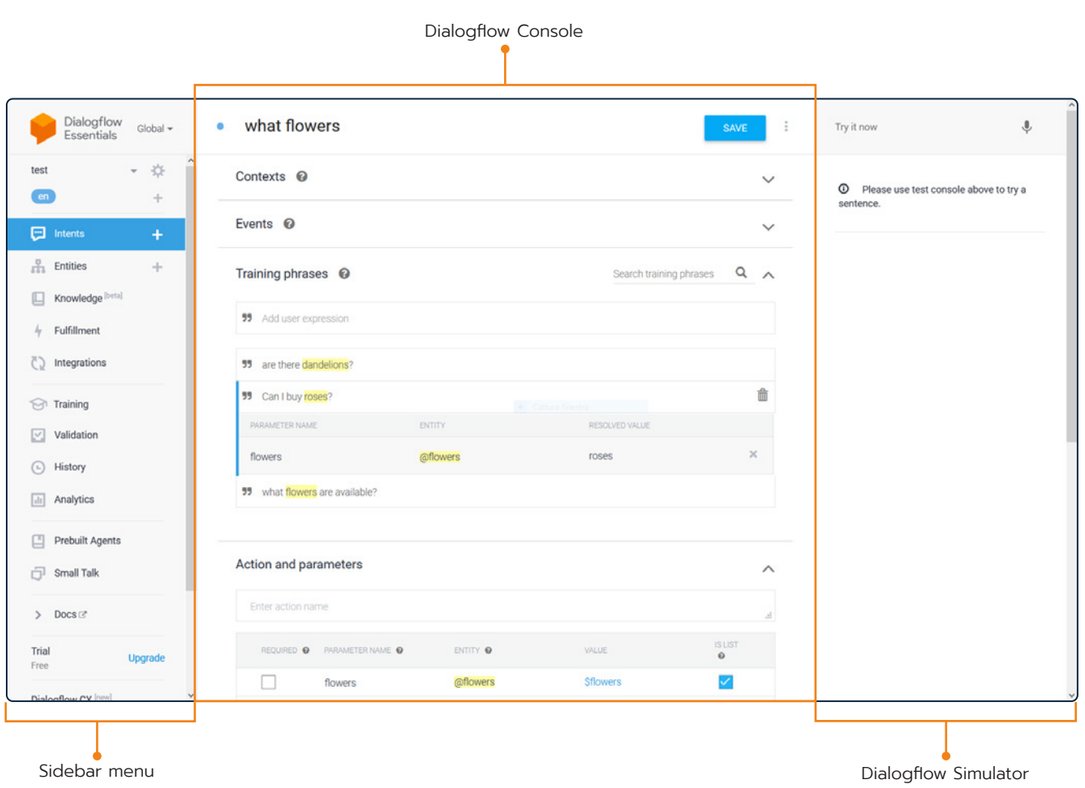
Entities
An entity is defined by an *entity type,* that is what kind of information the designer wants to extract from user input, and by a list of *entity entries*, that are the actual values that the entity can assume. Many of these entities are already defined like the ones about time, colours, locations or dates, but users can define their custom entities.

5 Dialogflow Console overview | Dialogflow ES | Google Cloud

Overview of the interface and interaction
To create the chatbot Dialogue flow provides a web user interface, called *Dialogflow Console*, where the user can build the conversational agent and test it with what in the system is referred to as *Dialogflow simulator*.

The interface is divided in three main areas: a sidebar menu on the left, where are listed all the tool tabs: intents, entities, integrations, training and so on. Next, in the middle there is the main content. In this panel users can view the content of each tab of the menu, and they can edit all elements and data needed to create the chatbot. The elements in each tab are organized with accordion sections. Each accordion contains one or more text input fields where the user is able to type the needed information. In the main content the user defines the elements that make up the chatbot structure.

Last element of the GUI is the *Dialogflow simulator* on the right. It provides a fast way of testing the chatbot during the development allowing to preview how the interaction will go down and to check for any errors. It has a text input at the top, where the user can type one of the training phrases to test the interaction. When the user types a sentence, the simulator will show the chatbot answer together with the information extracted from it, like what is the intent, or the parameters identified in it. The extracted elements function as links that, when clicked, open the corresponding tab. This is useful for the designer in order to rapidly access a tab and, eventually, make the necessary edits.



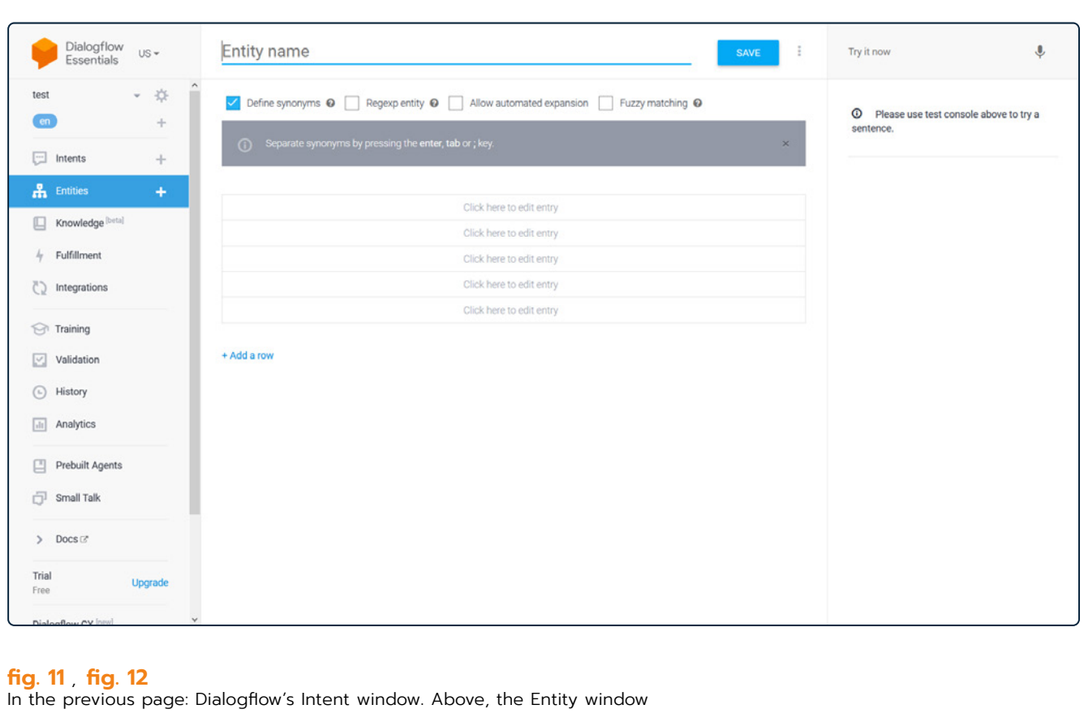


fig. 11 , fig. 12
In the previous page: Dialogflow's Intent window. Above, the Entity window

2.4.6. Amazon Lex

Implementation: hybrid
Context: general purpose
Intelligence: AI powered
Channel integration: cross-channel

Amazon Lex[6] is a service for building conversational agents that can support both voice and text inputs. It supports speech recognition for converting speech to text and NLU. Chatbot built with Amazon Lex can be used, for contact centre productivity or automation of simple tasks. Like Dialogflow it is cross-platform and supports integration with messaging platform such as Facebook Messenger, Slack, Kik and Twilio. To build the agent, the designer can choose to the Command Line Interface or use the graphical interface, called *console*. The designer must define the following key elements:

Intents
They are the action that the user wants to perform, for example ordering food. To define an intent the designer is required to provide an *intent name* that describes the meaning of the intent and must be unique. Then they must provide a list of *sample utterances:* various examples of how the user may convey the intent.

Slot types
They are pieces of data that may be necessary to fulfil an intent. Similarly to entities in other frameworks, to set a *slot type* the designer must give it a unique name and provide list of possible values that the chatbot will listen to.

AWS Lambda functions
The AWS Lambda functions can be configured to, perform validation of the data received from the user or fulfil intents. Lambda functions are pieces of code the developer can code in different programming languages, like Node.js Python Ruby Java or C#. This is a highly technical step that makes Amazon Lex the less suitable platform for non-programmer users.

Overview of the interface and interaction
The interface is structured in tabs. The editor tab, where the designer builds the chatbot, is structured similarly to Dialogflow interface: a sidebar menu on the left allows the user to switch form the definition of intents to that of slot types, at the centre there is the main content and, on the right, there is a *Test Bot*, a section where the user can try out the chatbot as they build it. Accordion sections are used to divide the main content of the tab and allow the designer to hide the input fields when not needed. *Training phrases* are here called *sample utterances*, but just like in Dialogflow, *slot types* are highlighted with different colours.

6 https://docs.aws.amazon.com/lexv2/latest/dg/what-is.html

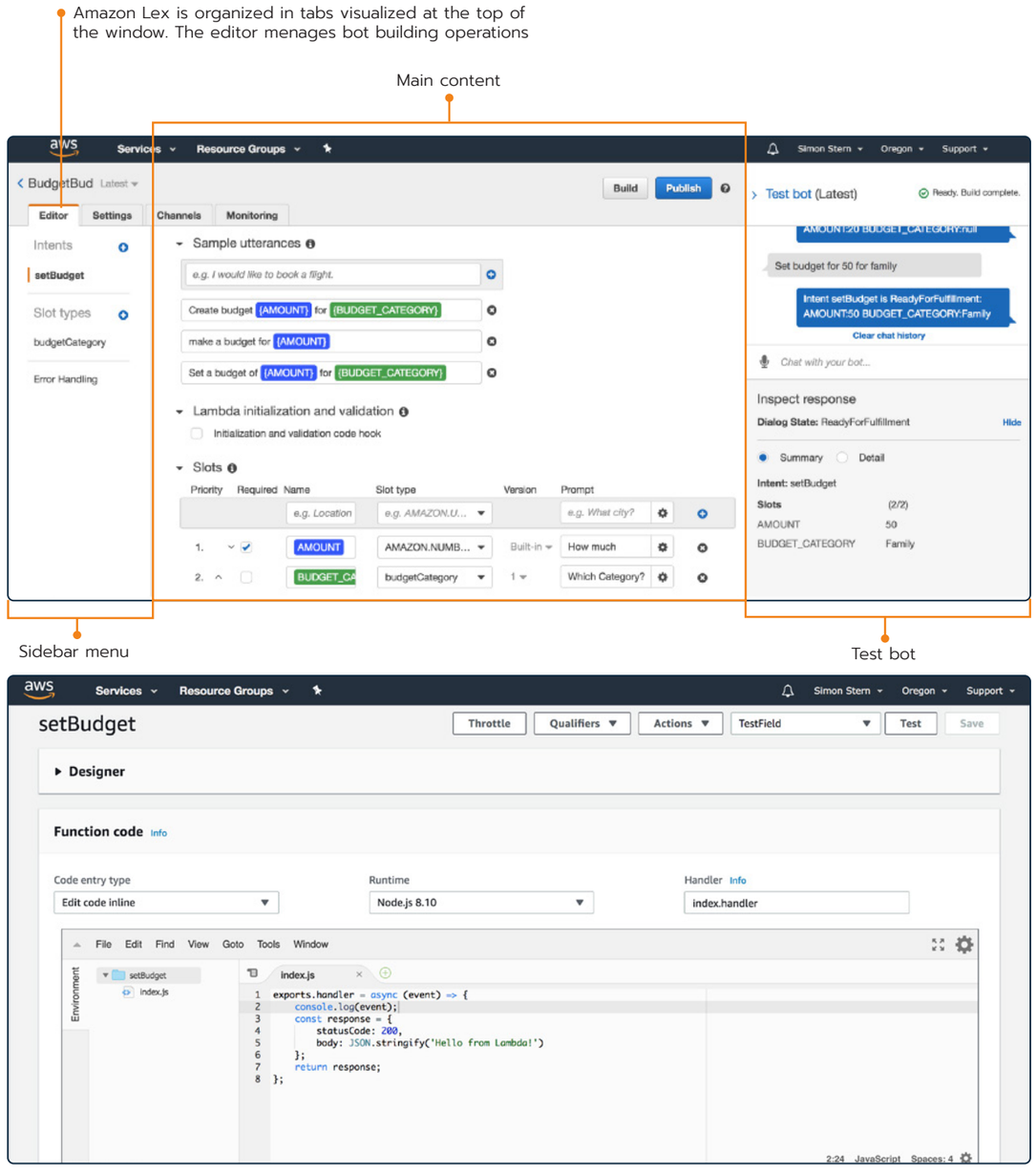


fig. 13, fig. 14
At the top, Amazon Lex Editor tab. Below, the lambda function window. Lambda function can be coded by the chatbot designer in different programming languages

2.4.7. IBM Watson Assistant

Implementation: hybrid
Context: general purpose
Intelligence: AI powered
Channel integration: cross-channel

In a survey by technology company Mindbowser (Mindbowser, 2017), IBM Watson Assistant[7] resulted to be the most popular platform to build chatbots, with 61% of the businesses in the sample choosing it. Watson Assistant is a tool built on a neural network by the same name, Watson. The service provides Natural Language Understanding and Processing to build intelligent chatbots either using one of the pretrained templates provided for common use cases, or by training a new agent with a custom database. The platform allows to create bot text based chatbots and voicebots and is cross-channel; users can deploy their chatbot on different social media platforms such as Slack, Facebook Messenger, WhatsApp or Intercom. Agent can be also used in call centres.

To build an agent, the designer is provided with a tool called *dialogue builder,* where users define the *conversation flow,* essentially the structure of the chatbot presented as a tree diagram. The tree starts always with a possible user utterance, while the following nodes in the diagram represent how the chatbot will respond following an *if-then* logic. The flow is made of *skills*. Skills are of two three types: *dialogue skills, action skills,* and *search skills*.

Dialogue skills
They contain the intents and entities needed in order to train the chatbot. When an intent is defined, like we have already seen, the user has to provide also a set of training phrases that could map to the intent. A *content catalogue* provides a set of prebuilt intents, such as greetings. In addition to that, the designer must provide also a *dialogue,* namely the response to give back once the intent is recognized. Chatbots built with Watson are context aware and can also ask more questions to clarify intents that have not been detected correctly.

For entities Watson Assistant provides predefined system entities, but the user can add new entities. Similarly to Dialogflow or Amazon Lex, to define an entity, users have to give it a name and list a set of possible values; Watson AI can expand the list with further recommendations. Entities values can be also defined as patterns, instead of giving a list of synonyms. For example, for an email entity the designer can specify just an expression that serves as an example of how an email address is structured, such as *text@text.com*. Like in Dialogflow words in intents can be tagged as entities so that they will be recognized.

7 https://www.ibm.com/cloud/watson-assistant

Action skills
They are the individual tasks that the chatbot is designed to help customers with. They could be described as blocks of interaction, that contain a series of steps that represent individual exchanges with a customer. Together such steps cover a task from start to finish. For example, an action could be telling a customer opening times of a business or greeting the user. As actions are composed of many conversation turns and can be reused in any project, once defined, they provide a way to speed up the building process of the chatbot.

Search skills
They are a type of skill that enables the agent to extract information from a configured data collection. To work, this skill interacts with another service by IBM, the IBM Watson Discovery service. The skill enables the agent to answer user queries that it was not designed for. In the skill slot the user choses form which collection perform the search and a set of possible utterances that the chatbot can use in different scenarios, such as when a search is successful or when no results are found.

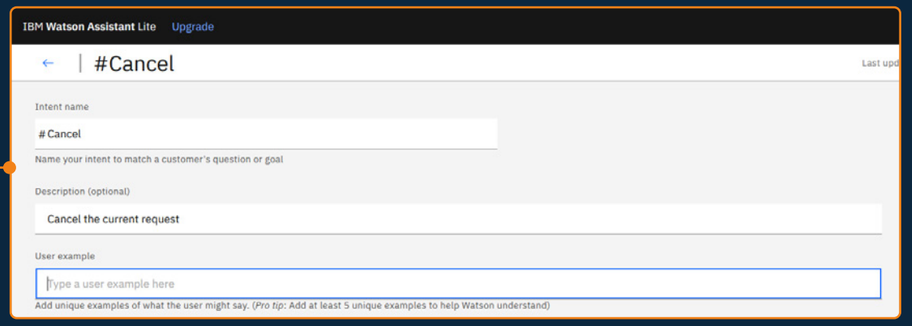
Overview of the interface and interaction
The user interface presents a sidebar menu and a main area. The main area is structured in four tabs: *intents, entities, dialogue, content catalogue*. In the intent and entities tab are very similar: they have on top the text field for the name of the intent or entity, respectively. After having set the names, user can start adding values and training phrases in the text fields below. The dialogue tab, instead, is where users can create the conversation flow, visually represented by a dialogue tree. Each node is a turn in the conversation and is designed as a card showing the name of the node, the intent it has to recognize, and the number of responses set. Clicking a node opens a pop-up window where users can add the intents, and the response to give back. In the header user can access through the *try it* button an interactive preview of the bot where they can test the chatbot as they work.

2.4.8. Flow.ai

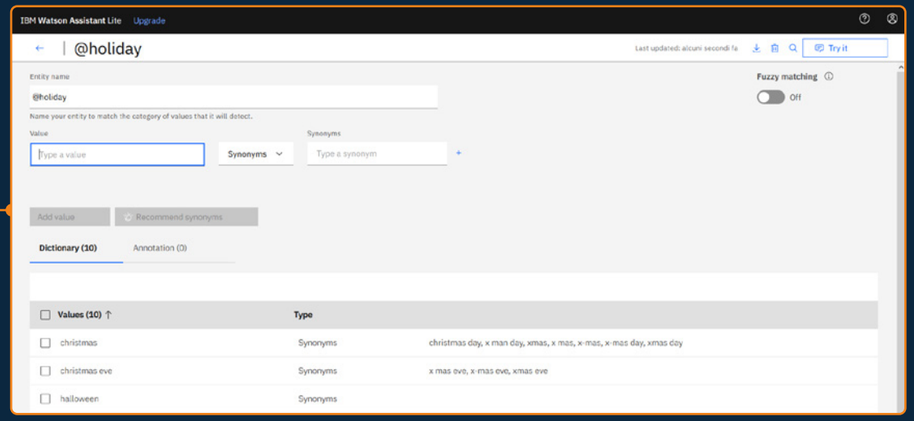
Implementation: no-code
Context: context-specific
Intelligence: AI powered
Channel integration: cross-platform

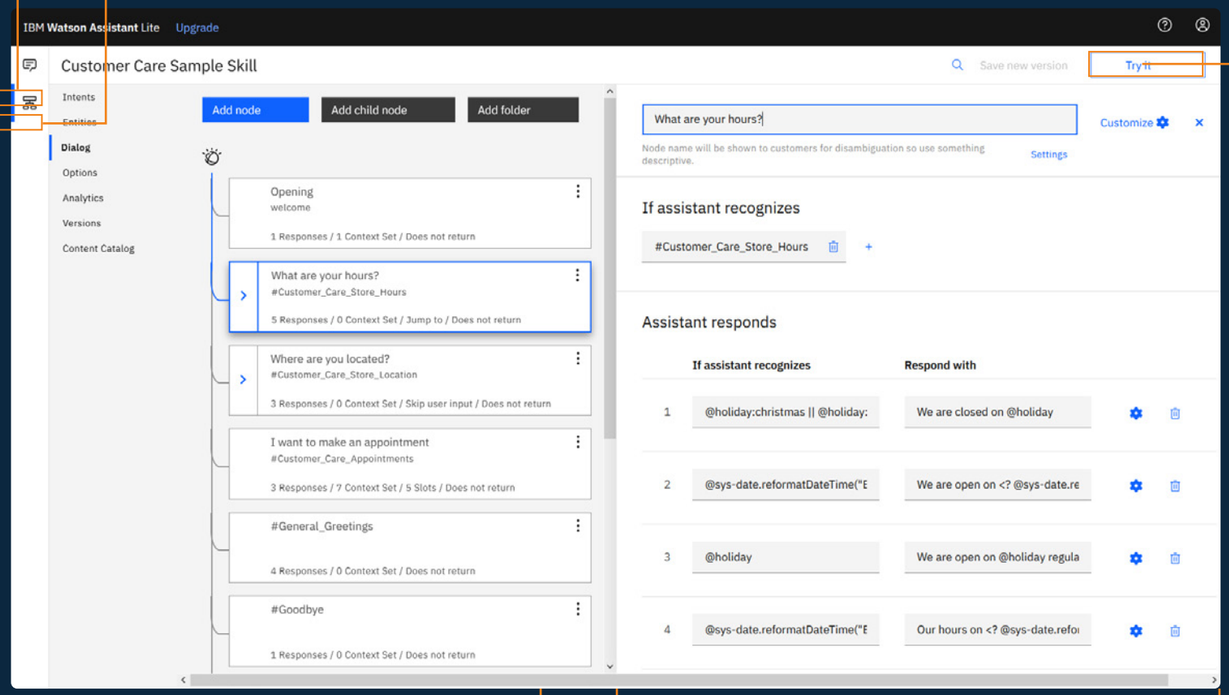
Flow.ai[8] is an online no-code platform that offers many templates for different use cases for marketing and customer service,

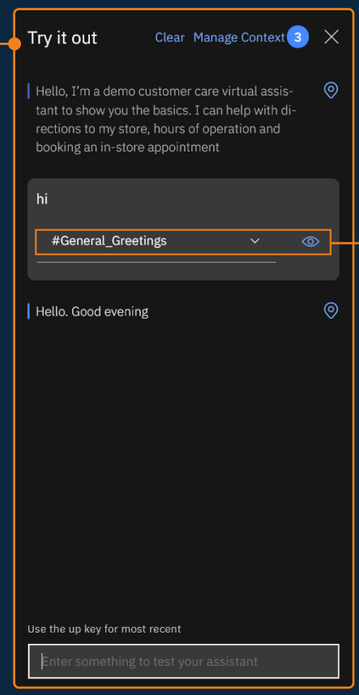
8 https://flow.ai/



In the entity window, users can use the dictionary provided by Watson or add manally new values







Users can modify the intent associated to the utterance as they try the chatbot

Dialogue window
Each node of the dialogue corresponds to a skill.

When a node is selected users can access the editor where they can specify the intent, responses and conditions that have to be met during the interactions.

fig. 15
An overview of the main components of the Watson Assistant user interface

although the user is not limited to use them. The platform has a proprietary Machine Learning engine that provides NLP to build AI chatbots. Chatbots built with Flow.ai are context aware meaning that the NLP engine keeps track of the state of the conversation allowing the chatbot to respond more appropriately to the user and remember data. The creation of the chatbot can be handled solely with the GUI, but for further customization the chatbot capabilities can be expanded with JavaScript.

The chatbot is built by creating a flow. Flows are visually represented in the GUI with a flowchart. A flow is specific for the dialogue the user wants to automate. Each flow starts with a *trigger*. A trigger is the user utterance. It can be a text or another event, such as sharing a picture or an audio. When a trigger is of the text kind, the user can classify it as an intent and specify more training phrases to train of the NLP model. In general, for each trigger corresponds a *reply* or an *action*. Replies are the answer to the user utterances. Flow.ai offers a rich variety of type of response: the agent can respond with just text a text message or give back a more complex answer that could include a location, carousels, video, audio or buttons. The combination of a reply and a trigger is called a step. Actions, instead, can trigger custom code or integrations. For example, an action can be used to send and retrieve data to and from a spreadsheet or connect with an email service.

Overview of the interface and interaction
The interface leverages the concept of flowcharts to visually represent the interaction between users and conversational agent. Each node in the flowchart is added with a drag-and-drop interaction and represents a step in the dialogue between users and agent, being it a trigger, a response or an action.
The interface is divided in three parts: On the left there is a sidebar menu where the user can access the flows, intents, entities and actions already created, and other functionalities. In the main area at the centre the user can view and build the flow by dragging the components (i.e., triggers, responses, and actions) from the right sidebar menu. Once a component is dragged form the menu it is added at a free end in the flowchart. The node of the flowchart shows a title input field to name the card and can show pictures and other media. Despite that, the editing of the content of each node is actually handled in the right sidebar menu, the last section of the interface. Once a node is selected the sidebar displays all the input fields of its editable properties. Lastly, at the top there is a header where users can find the *Try out* button to start a demo of the chatbot they are building.

2.4.9. Flow XO

Implementation: no-code
Context: context-specific

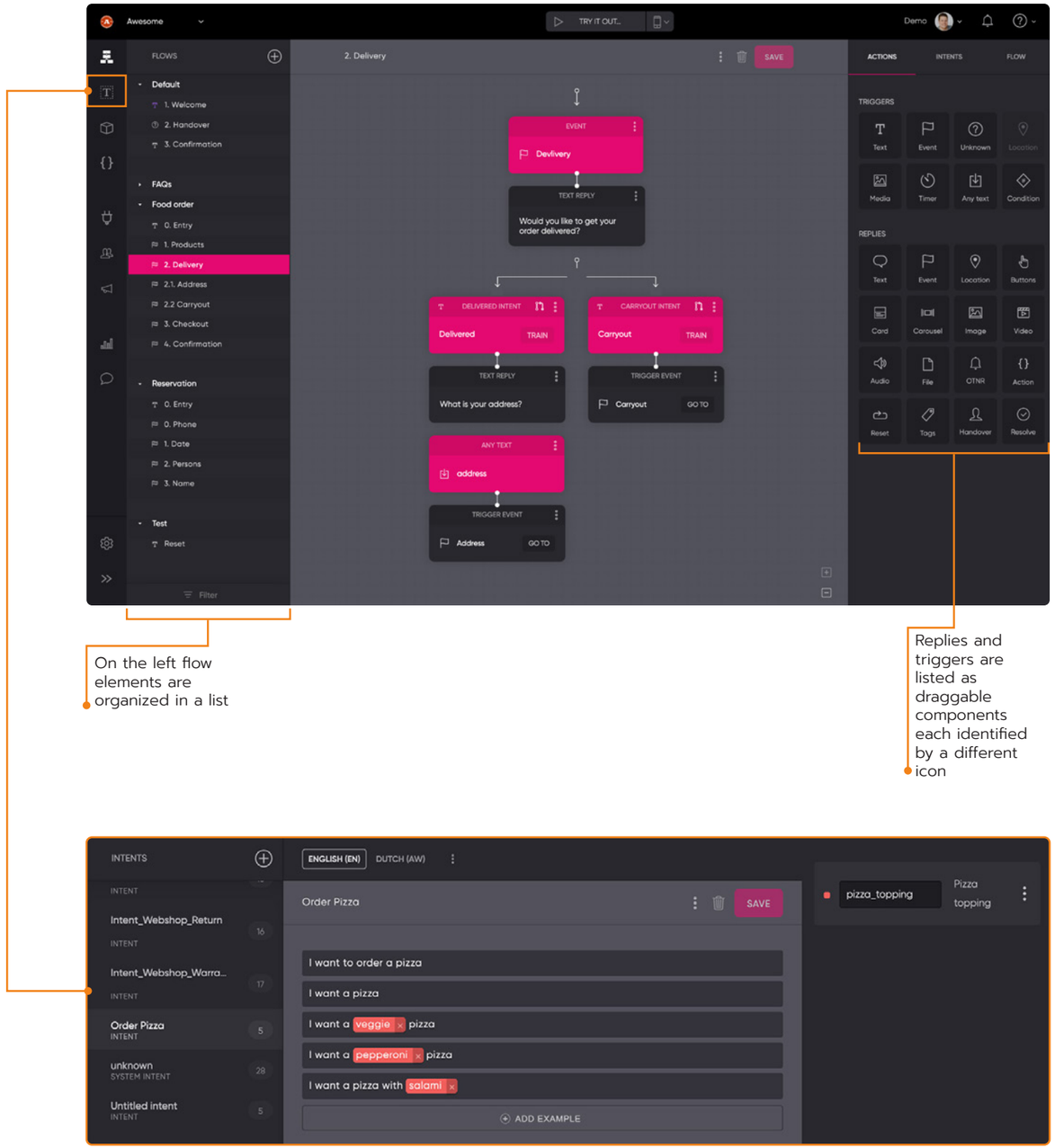


fig. 16,
An overview of the main windows of FLow.ai. At the top, Flow.ai flow builder. Below, a snippet of the intent window

Intelligence: rule-based, integration with Dialogflow NLU
Channel integration: cross-channel

Flow XO[9] is an online platform for the creation and hosting of chatbot aimed mainly at the creation of conversational agents for customer services and e-commerce, thus, provides a series of prebuilt templates for the most common use case like answering questions, applying to a service or book a room. Similarly, to the other platforms described, Flow XO is cross-platform. As such, it supports the connection to other messaging channels, like Facebook Messenger, Telegram, and Slack. Flow XO allows to create rule-based chatbots, but it can be integrated with Google Dialogflow to create an agent that uses NLU/NLP. In general, the platform relies heavily on third party integrations,

Flow XO provides a drag and drop interface where the user works with a visual representation, called *workflow (flow)*, of the conversation that the agent will have with a user. In the system a *flow* is essentially the script of the interaction between user and conversational agent. This script is represented as flow chart-like chain of *actions* performed by the chatbot in response to the utterances of the users that act as *triggers*. More specifically, flows are triggered by specific keyword in the user utterance. Usually, a bot is made of multiple *flows* That answer specific purpose in the conversation (i.e., booking a room). Each flow always starts with a *trigger*, that corresponds to any kind of input into the system. Most commonly this is a message from the user coming through one of the platforms that can be interfaced with Flow XO (i.e., Facebook Messenger). If it is a message, the designer must provide a list of possible words or phrases that the conversational agent will recognize as triggers and respond to, for example a list of greetings.

After a trigger is activated, an *action* is kicked off. *Actions* define the way the agent behaves in response to the user utterances. The system provides different kind of actions, like sending a message, asking a question or send an image.
From questions the chatbot can extract data like emails, names or dates. The saved pieces of data are called *attributes*.
The agent is able to store the data and use it in the course of the conversation, but also to add it to a spreadsheet.
The service, in fact, can be integrated with Google Spreadsheet so to have the ability to store and extract data.

Analysis of the interface and interaction
The user interface consists of three main areas and a sidebar menu where users can access the different tools and functionalities. In the middle there is a *workspace* divided in two parts: on the left it is shown the chatbot workflow, while on the right there is a live preview of the workflow structured like a

9 https://flowxo.com/

common chat interface. This way, they can keep track of all the questions and text written into the *action* modules.
The workflow develops vertically down the left side of the page. *Action* and *triggers* are visually represented by a card that contains a pictogram specific for the different kind of system actions, the name given to the action, an edit button and a filter button. Through the edit button the designers access a pop-up window where they can set the triggers and the outcome of the action.

The whole building process is segmented in a string of modals that guides the user through each operation.
To actually test the chatbot and see if the user input is parsed correctly, Flow XO provides also a *Test Console*, a messaging application emulator, that can be called from the header. In the *Test Console* the designer can impersonate the end-user and try the agent for themselves.

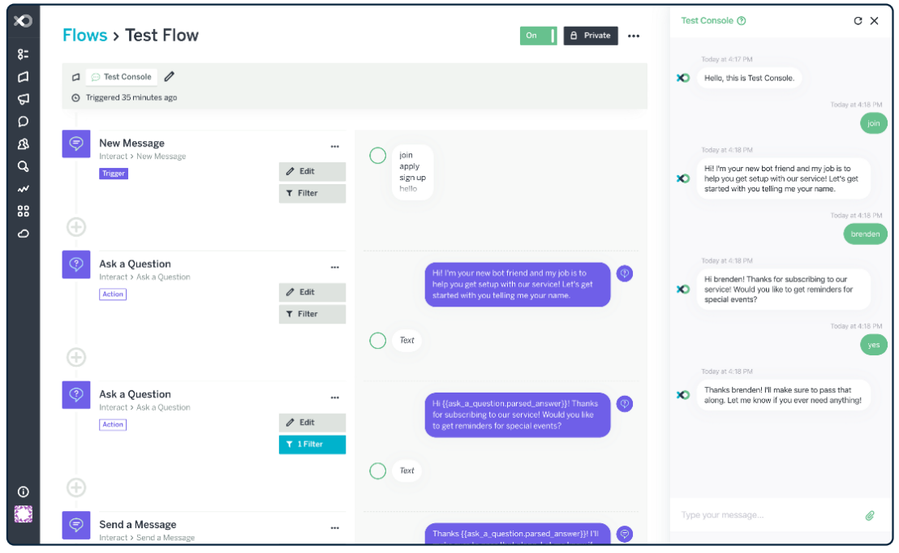


fig. 17
Flow XO workspace. Right to the tree diagram of the flow, intents and responses of each module are arranged following a text messaging platform layout

2.4.10. ManyChat

Implementation: no-code
Context: context-specific
Intelligence: rule-based
Channel integration: single channel

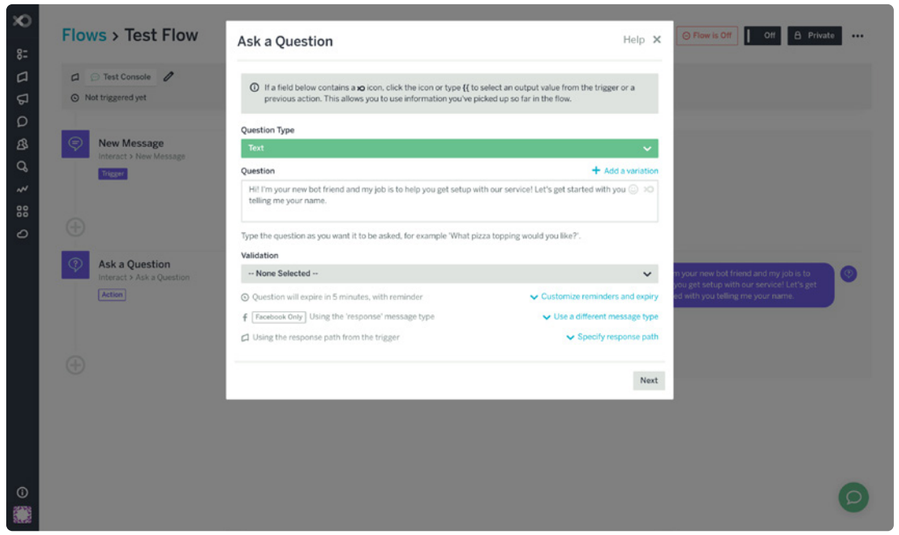


fig. 18
In FlowXO modal intents and responses are set in a modal window. The content of the modal depends on the type of trigger used for each intent. In the picture we can see the *Ask a question* trigger.

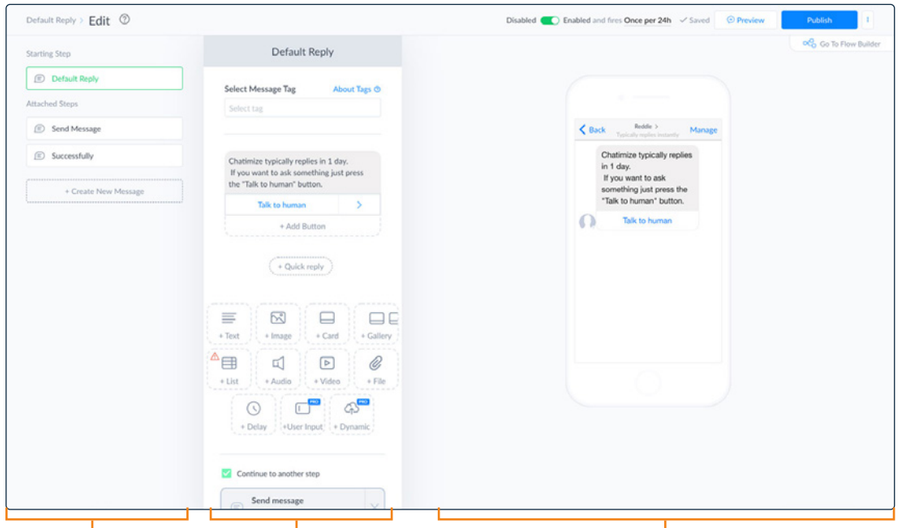
ManyChat[10] is a chatbot platform for the development of conversational agents specialized in marketing and CRM activities; different templates are available that cover the most common activities from lead management to e-commerce functionalities. ManyChat is a single channel platform where the bots can be published only on Facebook Messenger, but messages can be broadcasted also via SMS, although it is not a channel where the end user can interact with the bot.
The platform does not support NLP and allows to build only rule-based chatbots. To build the chatbot, the user defines *action rules* and *keywords*. ManyChat provides two different interfaces: the Basic Builder and the Flow Builder. The Basic Builder is a compact editor where the messages that make up the chatbot dialogue are presented in a linear fashion. The Flow Builder, instead, adds a flowchart diagram representation of the dialogue like we have seen in Flow.ai.

Every project is called *flow* and starts with a user input that contains a keyword that triggers the flow. Keywords are associated to rules, like *message is, message contains,* or *message begins,* that allow the user a certain degree of control on how the message will be interpreted. When defining a keyword, users define also the chatbot response. The response in general is a text paired with a set of buttons that list the possible further interactions with the bot. Responses can be enriched with media, though *blocks,* buttons that add images, video and other media to a response. To understand user input the response must contain a *user input block,* that can parse the user answer and understand if there is a recognizable keyword in it. Actions rules, instead, are used to perform an action, like sending a confirmation email or notifications, when certain *triggers,* or conditions, are met. To set up custom ones, users have to define a trigger; it can be a date, or a more complex condition such being a new customer.

Analysis of the interface and interaction
The Basic Builder and the Flow Builder offer two different interaction paradigms that can be useful in different scenarios. The Basic Builder (fig. 19) offers a linear view of the conversation flow more suited for simpler bot. The user interface is divided in two different areas. A sidebar menu on the left allows the user to change window and access the other tools, like the dashboard window or the templates window. The second area is the main content that is structured in three sections. On the left there is the list of the messages created, at the centre there is the editing section, where the designers can edit the messages they create by adding text and blocks. Lastly, on the right there is a live preview of the chatbot.

The Flow Builder (fig. 20) has a similar structure despite having a different interaction paradigm. The main area is now used as

a workspace where users can build the chatbot as a flowchart. The fist node is added from the sidebar menu by clicking the *welcome message* button, while the next nodes are added automatically or by clicking the *choose next step* button at the bottom of each node. The nodes show the content of the response and some statistics about the node. By clicking on a node, it is possible to edit it using the same editor present in the Basic Builder. Lastly, to access the preview, they have to click the *preview* button on the right of the header.



On the left dialogue elements are organized in a list

Each element can be edited here by adding blocks. Blocks contain the chatbot possible actions, such as giving an aswer or show multimedia content

Each element can be edited here by adding blocks. Blocks contain the chatbot possible actions, such as giving an aswer or show multimedia content

fig. 19
The Basic editor

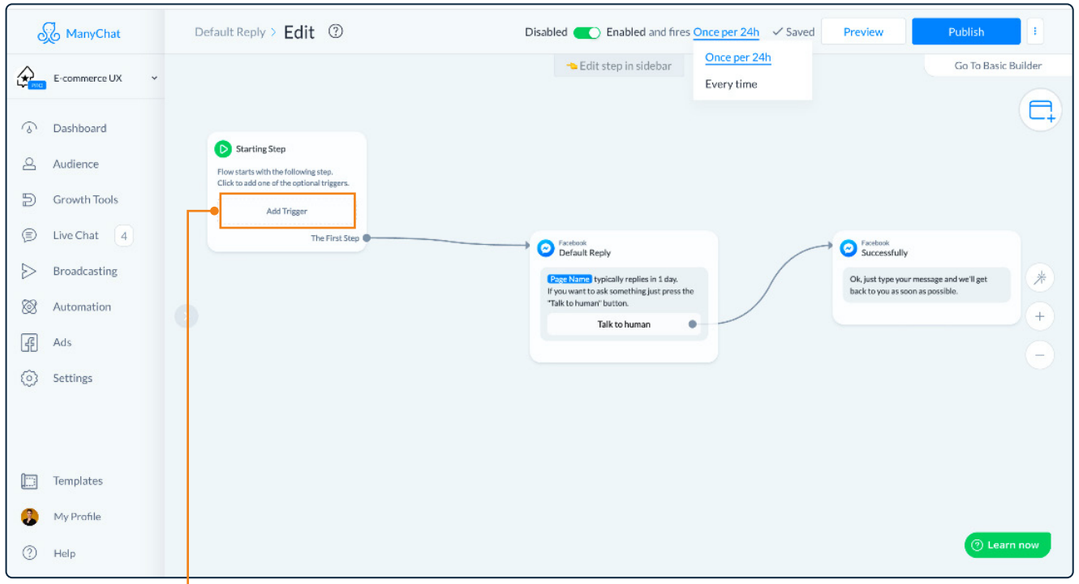


fig. 20
The Flow editor. To edit the flow, users must click on one of the nodes to open an editor similar to the one seen in the basic builder

2.5. The Gap in the State of the Art

According to the current practices, chatbot development requires a large body of training data obtained from collecting examples of user dialogues. These data are expensive to gather and are difficult to reuse across different domains (Yaghoub-Zadeh-Fard et al., 2020). Additionally, these approaches currently lack integration between the conversation system and the database from which data are extracted. As we have seen, in fact, the platforms currently available focus on manual intent and entity definition, with integrations with databases being absent or limited to the importing of data for entities from CSV files, as in the case of Watson Assistant or Dialogflow. This implies that several data characteristics, which are relevant – if not necessary – to manage the conversation and to train the dialogue system, must be provided manually, even if they are already implicitly available in the database model. A critical challenge, therefore, lies in understanding how to make the development of chatbots for data exploration scalable and sustainable, especially in terms of design models and methodologies (Akcora et al., 2018) (Pereira & Díaz, 2018).

This work capitalizes on a chatbot design framework, CHATIDEA, which proposes a set of modelling abstractions to identify key data elements in a relational data source, and maps them on user utterances to make conversations for data exploration possibles. Upon this framework is built an interface, that unlike those seen until now, bases the design of the chatbot upon the structured data of a relational database. The interface proposes a novel interaction paradigm with respect to those seen in this chapter. First, because it integrates database visual language to present the data. Secondly, because it proposes a novel conversational paradigm for the generation of intents. With this paradigm the intents are presented in a chat-like fashion, simulating a semi structured conversation with the end user.
This is possible because the relationships existing between the different tables offer a sort of scaffolding of the conversation that allows the user to have a sense of context when annotating the intents, as they are not presented as just a list as in platforms like Dialogflow or Amazon Lex, but in a broader conversational context. Furthermore, again thanks to the nature of relational databases, the conversational paradigm offers a more compact and straightforward way to organize the conversation branches as they are tied to the data. One of the drawbacks of flow-chart styled interfaces, as Flow.ai or ManyChat, is, in fact, the difficulty in managing complex conversational agents with many conversation flows branching on different topics.

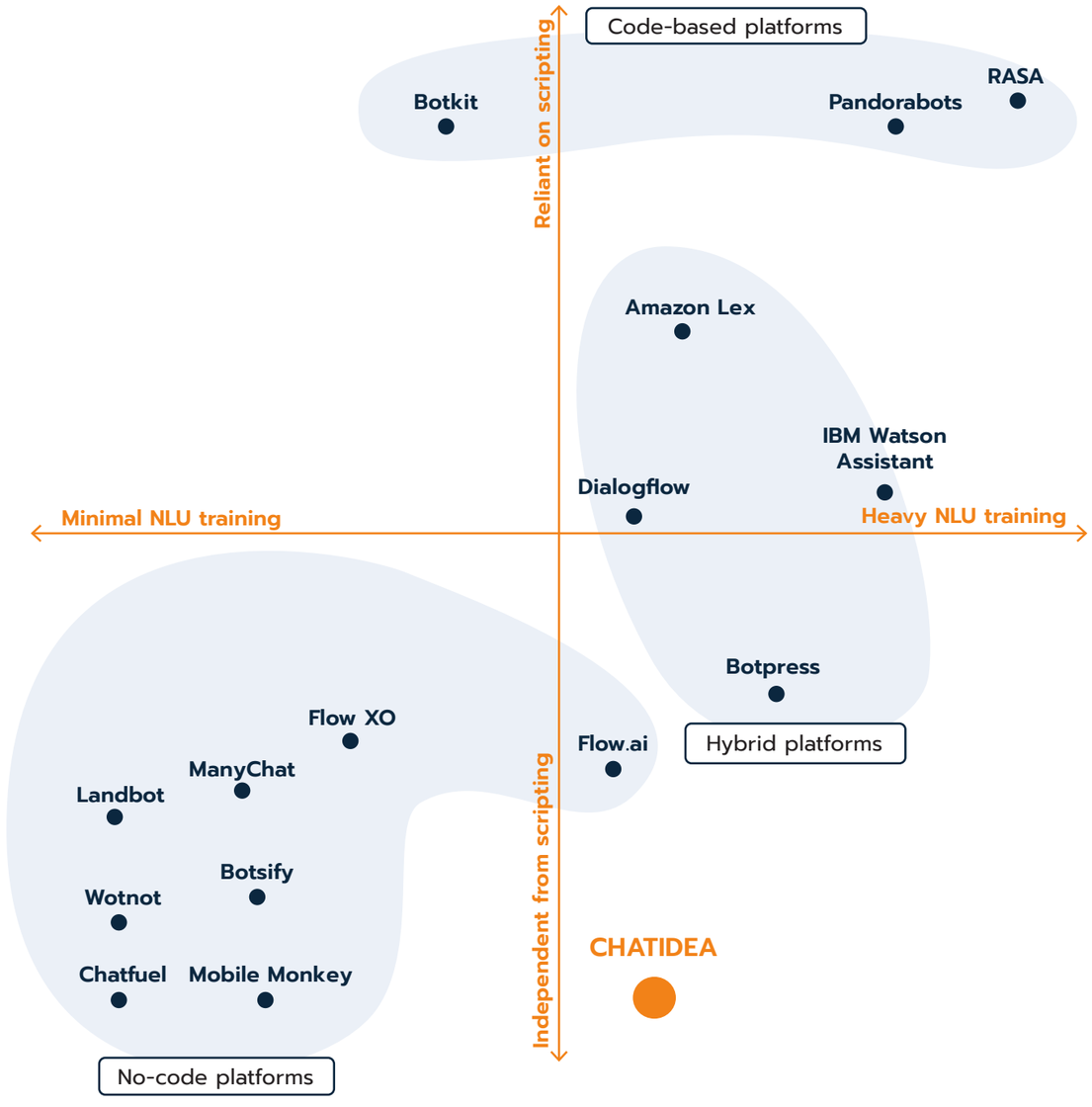


fig. 21
Quadrant diagram showcasing the distribution of the platforms analysed in table 2

3. CHATIDEA

3.1. Summary

This thesis capitalizes on a *framework for the rapid prototyping of chatbots for data exploration* (Castaldo, 2019), CHATIDEA, that allows the development of such chatbots and presents a graphical user interface to aid developers in the generation of a set of files, called *Database Schema Annotations,* that are needed for the mapping between data and end-user utterances.

3.2. Relational databases

The framework is developed on the basis of relational databases. A database is collection of structured data. A relational database is a database where the data is organized in tables, called relations, that are collections of data of the same type (Whitehorn & Marklyn, 2007), for example a collection of students or items on sale. The tables are made of rows and columns. Columns are also called *attributes*. Each attribute is different and stores a specific kind of data, such as first names, surnames, age, and so on. Rows, also called *records,* must be different from one another and each of them represents an entry in the table. So, in a table about customers each row represents a different customer.

Another characteristic of relational databases is that tables can be logically connected by relationships (Whitehorn & Marklyn, 2007) where the first table is the *referencing* table and the second is the *referenced* table. The relationship exists when an attribute in the referenced table is present in the referencing one, thus "linking" them. Such attribute is called *primary key* in the referenced table and takes the name of *foreign key* in the referencing table. The primary key is an attribute that has the characteristic of being able to uniquely identify every and each row of the two tables. Furthermore, every table must have a primary key, while foreign keys is present only when a relationship exists between two tables. Let us now see an example.

Suppose that we have a database about students enrolled at Politecnico di Milano and there is a table storing personal information about the students called STUDENTS and another with their contact information, called CONTACTS. As we have said, a primary key is an attribute that can univocally identify

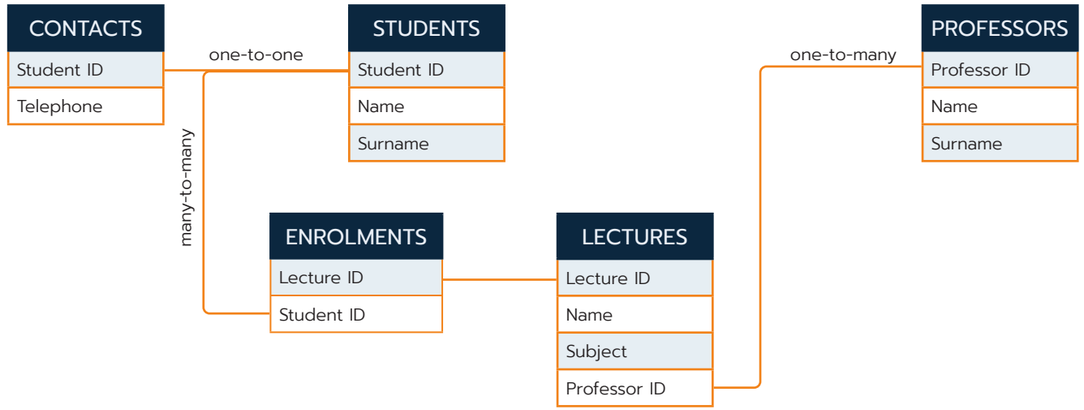


fig. 22
An example of a relational database and the type of relationships between them

each row in a table. In the case of the table STUDENTS, the attribute that can identify each student univocally is their student ID, so that can be a primary key. If we want to create a relationship between the table STUDENTS and CONTACTS in order to know which student has which contact information, the primary key of STUDENTS (i.e., the student ID) must be repeated in CONTACTS, thus creating the link between their contact information and the students themselves. When the primary key is referenced in the table CONTACTS it is called foreign key, as it essentially belongs to the other table, STUDENTS.

Lastly, it is important to note that relationships can be of three types: one-to-one, one-to-many and many-to-many. Two tables have a one-to-one relationship if one row from the first table corresponds only to a single other row in the second one. Continuing with our example, one student can correspond to only one contact record and vice versa. Instead, we have a one-to-many relationship, when to one row from the first table correspond many rows in the second one. That is, one professor can teach many classes, but all the classes are held by the same professor. Lastly, we have many-to-many relationships when multiple rows in the referencing table are associated with multiple rows in the referenced table. For example, students follow many lectures, and each lecture is followed by many students. Unlike the other two types of relationships, a many-to-many relationship cannot be represented directly with only the two tables. and needs to be broken down into two one-to-many relationship. To do so, a third table is used acting as a "middleman." This table, called *join table* or *bridge table*, will then contain both primary keys of STUDENTS and LECTURES plus a third primary key used to create a univocal link between lectures and students. In our example such table could be called ENROLMENTS where we have three attributes: the two foreign keys from STUDENTS and LECTURES plus an attribute

with enrolments numbers. This way, each pair of students and lectures is identified by one unique value.

3.3. Structure of CHATIDEA

CHATIDEA is presented in two previous works. It is first presented in the Thesis *A Conceptual Modeling Approach for the Rapid Development of Chatbots for Conversational Data Exploration* (Castaldo, 2019). Then, it is further expanded in the thesis *Designing and Validating Conversational Agents for Data Exploration* (Ferreri & Notari, 2020). The procedure for chatbot development described in this works is semi-automatic. In fact, while the majority of the steps necessary to generate the chatbot are automatic, some are not and require the work of what is defined as a *Chatbot Designer* (Castaldo, 2019). As we

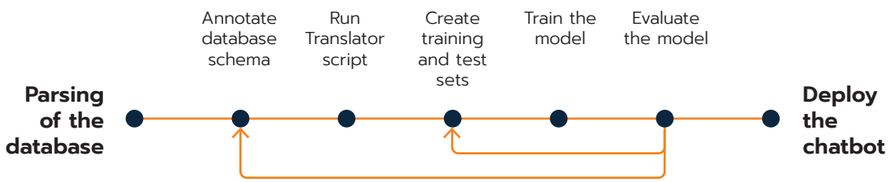


fig. 23
A diagram of the steps necessary to generate the chatbot
(Ferreri & Notari, 2020)

can see in figure 23, the steps for the generation of the chatbot are essentially five, with two additional steps, evaluation and deployment, that end the pipeline but happen after the actual generation.

The first step is the *parsing of the database schema (fig. 23)* and it is automatic. The framework focuses on relational databases. After the parsing, a simplified version of the database is obtained, called schema. In this version the structure in terms of tables, attributes and relations among entities is highlighted. This step is necessary for the following one: *Database Schema Annotation*. It is the main abstraction and may also be regarded as the core of the process as it is fundamental to the mapping between the data and the query the end-user will express while interacting with the chatbot. Moreover, this phase is the main manual phase that the Chatbot Designer must perform. The designer tags the database schema with keywords that will be necessary to generate the conversation. Not only, but the annotation step is also what makes the system able to convert the natural language request into a query in SQL and to give back the answer in natural language as well. Next the designer

runs a script called *translator;* this component is needed to create the modelling files necessary for the fourth step in the process, the creation of *training material*. The training material is obtained thanks to the model previously created. The model is used to generate a set of phrases based on the database and the annotations written at the start of this procedure. Lastly, there is the *training phase*. It is done by running a script called trainer and results in the extraction of the NLU model needed to understand the user utterances (Ferreri & Notari, 2020). Below (fig. 24) an example of interaction between chatbot and users is reported.

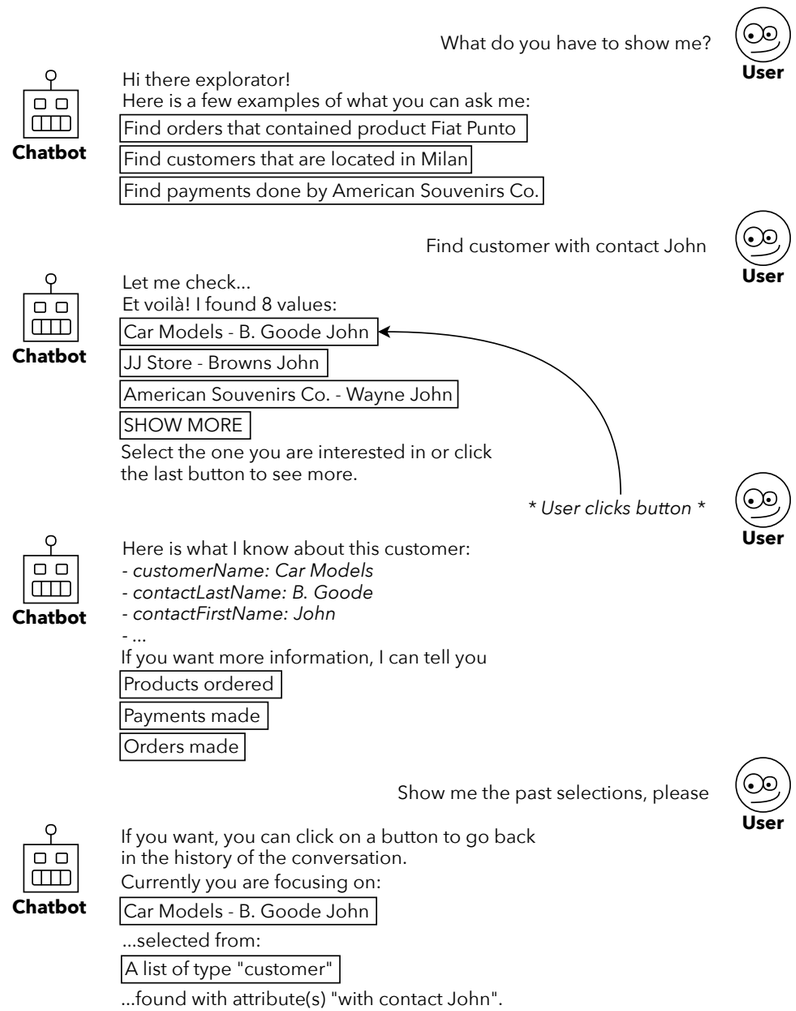


fig. 24
(Castaldo, 2019)

3.3.1. Annotation of the Database Schema

We will now see the technical details of the annotation step. In the current version of the CHATIDEA framework, to generate the mapping between the contents of the database and what can be understood by the chatbot, three JSON files have to be written by the chatbot designer: a *Schema*, a *Concept file* and the *Display Annotation file* (Ferreri & Notari, 2020). The three JSON files are structured as follows.

The *Schema file* contains for each table of the database the attributes, primary key and, if present, references to other tables. The information contained in the *Schema file* does not add meaning to the data in regards with the conversation, but is fundamental for the description of the database. The *Concept file,* instead, contains all the annotation concepts that enable the chatbot to match the data with the natural language phrases used during a conversation. Lastly, the *Display Annotation file* specifies what data will be shown to the user and what will not. In a database, in fact, there can be data that should not be disclosed to the user or that are useless in the conversation. To solve this problem, in the *Display Annotation file* the chatbot designers will select and filter the attributes they want to be shown to the user during conversation.

```
"location_edifici": {
        "column_list": [
            {
                "attribute": "id_edificio",
                "display": "Building ID"
            },
            {
                "attribute": "nome_edificio",
                "display": "Name"
            }
        ]
```

listing 1
An extract from the Display annotation JSON file.

Together with that, they will also provide a name to introduce the actual values of each attribute. For example, if a table contains an attribute for peoples first names, when presented to the user, the attribute about first names could be introduced by the word "name", so that the end user will see *name: Mario*. These are all the annotations that must be defined in the *Concept file* for each table of the database:


```
"location_edifici": {
        "column_list": [
            "id_edificio",
            "old_idsede",
            "nome_edificio",
            "codicepatrimonio_edificio",
            "ordine_edificio"
        ],
        "column_alias_list": {
            "id_edificio": "building id",
            "old_idsede": "building old id",
            "nome_edificio": "building name",
            "codicepatrimonio_edificio": "codice patrimonio",
            "ordine_edificio": "order"
        },
        "primary_key_list": [
            "id_edificio"
        ],
    "references": []
},
```

```
"element_name": "building",
        "aliases": ["buil ding", "buildings"],
        "type": "secondary",
        "table_name": "location_edifici",
        "show_columns": [
            {
                "keyword": "",
                "columns": ["nome_edificio"]
            }
        ],
        "category": [],
        "attributes": [],
        "relations": []
    },
    {
        "element_name": "room",
        "aliases": ["rooms"],
        "type": "secondary",
        "table_name": "location_locali",
        "show_columns": [
            {
                "keyword": "",
                "columns": ["descrizione"]
            }
        ],
```

listing 2, **listing 3**
At the top, an extract from the Database Schema file . At the bottom, an extract from the Concept file

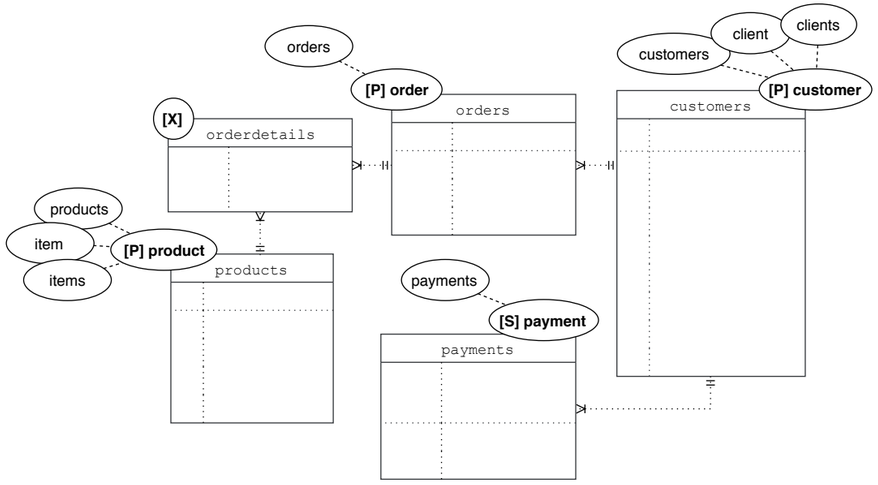


fig. 25
An example database schema. In the balloons on each table some alias have been specified (Castaldo, 2019)

Conversational Object
It is a keyword, meaning a word or an expression, used to identify univocally each table. To allow the user to refer to tables in different ways and still be understood by the conversational agent, a list of synonyms, called *aliases,* can be provided for each table. So, if in the database there is a table storing information about customers (fig. 25), as a conversational object it could be used *customer,* while some aliases could be *client, person,* and their plural counterpart.

Type
Defines the role that the table plays in the exploration of data. Each table can be assigned one of these three types: *primary* ([P] in fig. 25) if the information therein can be understood and represented without a context, such as a table containing the personal data regarding the clients of a shop. Another type is *secondary* ([S] in fig. 25). It is used when the information stored in the table is so strongly related to other entities that it requires previous information to be understood. Continuing the example from before, showing the purchases made by each client is meaningful if the user has already searched for the client, as who bought an item is what makes each purchase relevant. Lastly, a table can be classified as *crossable* ([X] in fig. 25) if it is a bridge table. This kind of tables bears no relevance to the conversation, but is used during join operations[11].

Display Attributes
These are one or more attributes that are used to represent each element when returned after a search. They are particularly useful when dealing with lists, as the limited information to be displayed makes it easier for the user to scan the answer.

11 A join is an operation that combines data from different tables in a relational database. They are possible if a relationship exists between them.

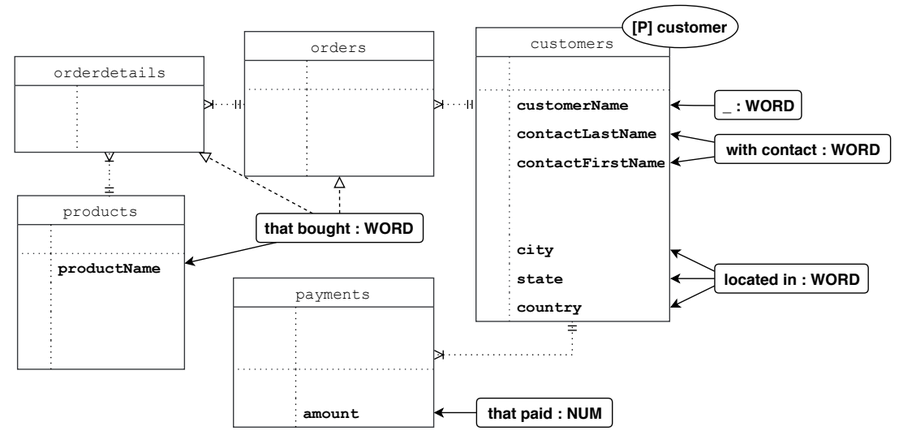


fig. 26
In the rectangular lables we can see an example of the conversational qualifier annotation, with a focus on customer. (Castaldo, 2019)

For example, when returning a list of customers, each of them can be represented by name and surname only.

Conversational Qualifiers
A search like *find customers* would return all the instances contained in the table *customers*. If the user wants to perform a selection, such as *"find customers located in Milan",* the chatbot needs to be able to understand that in the answer it should show only the customers that live in Milan, i.e., that *located in* is a phrase used to select customers that have *Milan* as the value of the attribute *city*. This is achieved by defining and associating the *Conversational Qualifiers*, keywords or phrases like *located in,* to the attributes on which the user would want to perform a selection (in this case the attribute *city*). The attribute can belong also to another table that has a relation with the one searched by the user.
For the utterance to be correctly interpreted by the chatbot, the designer must also specify the *Conversational Type*, which defines the type of data contained in the attribute tagged with a conversational qualifier. There are four types of data:

WORD: any information that does not have a particular structure or syntax
NUM: numerical values
DATE: Dates and time values
Custom ENUM: entities that can assume enumerable values[12]

In the case of *Milan,* the Type will be WORD as it is just a name. Lastly, the real value of an attribute tagged with a conversational qualifier is called *Conversational Value* (i.e., Milan is one of the values of the attribute *city*). So, going back to the previous example, in the sentence *find customers located*

12 An enum is a special "class" that represents a group of constants. In general, it's used when you know all the possible values of a variable (McConnell, 2004)

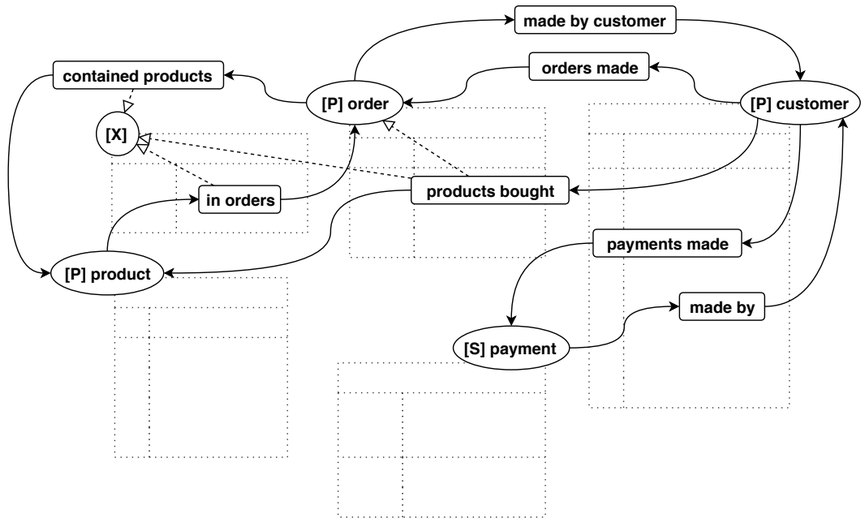


fig. 27
Conversational relationship annotation, made with respect to the customer (Castaldo, 2019)

in Milan, located in is the conversational qualifier, *Milan* is the conversational value of the attribute *city* and its conversational type is WORD as it is a simple name.

Conversational Relationships
They are keywords that identify a relationship between two tables. With this annotation the designer defines the name to be used to refer to a relationship during conversation. It could be the name already present in the database schema, or it could be a more meaningful one for the context of the conversation. In the context of the conversation, conversational relationship act as navigation suggestions to help the user in the exploration of the database. For this reason, relationships between tables are presented as buttons at the end of an answer.

Categories
This annotation is done only for tables tagged as Primary. If the table contains one or more attributes whose values can be used to categorize the rest of the data, then the attribute in question can be used to display the data in a pie chart. The designer will have to select the attribute that will be used for the data visualization and provide an alias for the attribute that will act as the legend helping the user interpret the chart. For instance, if we want to visualize how the customers are geographically distributed, we could choose to tag the attribute *city* as a category and give it the alias "geographical distribution."

References
By design, in a relational database, two tables reference each other by a foreign key and a primary key. Since they may not

always have a meaningful value, references were added: with this annotation, the chatbot designer is able to choose another attribute more meaningful and representative from the table to show the user instead of the primary key. For example, the table *customers* references the table *purchases*. Technically, if the user asked "*show me customers' purchases*" the chatbot would answer with the foreign key. Instead by adding a reference the chatbot designer can choose to show another attribute such as the type of product purchased.

4. Methodology For Dialogue Design

4.1. Summary

Starting from an analysis of the technical and theoretical resources regarding CHATIDEA, in particular the JSON files, it was possible to extrapolate a series of requirements and issues that had to be taken into account during the design, and that conditioned the final result. This chapter will explain the design process behind the graphical user interface here proposed for the CHATIDEA framework; first, by explaining which where said requirements and issues. Then, the reasons behind the choice of a conversational paradigm as the preferred interaction paradigm will be addressed, together with some background that introduces said paradigm.

4.2. Design Requirements

The design process started by identifying the fundamental requirements of the interface. As a first requirement, the interface had to support a visualization of the database, not only for annotation purposes, but also in order to give the users a way to have a quick overview of the entire database they are working on. That way even users not perfectly familiar with the data could always rapidly look up the data and find the piece of information they were looking for. To visualize the database, a relationship graph was chosen, for it is one of the most common way to represent databases, and it allows to show the tables together with the relationships that link them.

The second requirement was designing an interaction paradigm that could support the annotation procedure without having an in-depth knowledge of all the concepts of the database schema annotation files. Such a paradigm had to be intuitive enough to not depend on an explanation of each annotation concept for the annotations to be correctly defined. A linear listing of intents and entities as seen in Dialogflow and Amazon Lex, was considered to be too confusing for this framework, especially since the users work with large body of data. For this reason, it was important to find a paradigm that could also visualize the intents into the context of a dialogue structure, as seen in Watson Assistant or other flow-chart-based platforms like Flow.ai. Since a relationship graph was already in use to represent the database, a flowchart as seen in platforms like Flow.ai and ManyChat, would have been confusing, as the visual representation of the data

and conversational elements would overlap. Thus, by applying a recursive logic, the choice fell on a conversational paradigm so to, that will be described later in the chapter, so to design a conversation with a conversation.

Furthermore, the design also presented two particular challenges: first and foremost, getting around the repetitiveness of the annotation process so that there would be no ambiguities while working on the annotations. During the process, in fact, the chatbot designer is called multiple times to rename the same attributes, each time for a different goal. Renaming of the attributes happens, for the first time, when generating the database schema to give the data more readable names. Then, tables are renamed for the conversational object annotation, while single attributes are renamed for the display annotation. Lastly, attributes are renamed once again for the category annotation to create the legend accompanying the data visualization. This challenge was particularly important because the repetitiveness of the renaming operation may lead the user to be tempted to skip the renaming operation after doing it the first time it is asked, thus possibly posing a problem for the generation of the files. For this reason, in order for the user to fully understand the need for each naming operation, the various renaming tasks have been differentiated both by visual means and by carefully curating the message introducing each task, as we will see later in the chapter.

The second challenge was differentiating between annotations that help the chatbot interpret user utterances and those related to conversational relationships, which are the annotations that manage the generation of exploration suggestions by the chatbot itself to the end user. The problem is directly linked to the need to make the interface as independent as possible from specific knowledge of the framework and is one of the reasons that lead to the conversational paradigm used to design of the graphical user interface. The remaining of this section illustrates the different concepts to be managed for annotation generation, and the related strategy adopted for their definition by the end users.

4.3. The Conversational Paradigm

Conversational design is the practice of designing device interactions so that they resemble conversations between two people (Leong, 2004). Using the principles that make interactions between users productive, conversational design aims at making human-computer interactions more intuitive and familiar for users. Conversational design focuses on two main objectives. The first is developing conversational interfaces like chatbots, that can interact with users using natural language and can

exploit specific characteristics of conversations, such turn-taking, feedback and being able to respond to both non-verbal and verbal interactions (Cassell et al., 2000). A second objective regards the use of features extrapolated from human-human interactions to develop more intuitive user interfaces (Nickerson, 1976). Such rules include giving feedback to users to give a sense of presence when the system is busy performing tasks.

As of now, the rise in popularity of messaging platforms has made conversational user interfaces one of the most common kinds of user interface to interact with (McKitterick, 2016) leading some to prospect that this type of interface will become the future of User Interface design (Vanhemert, 2015). This also means that conversational interfaces like messaging chats are the most familiar for users. Due to this particular property, to present the annotation concepts, the UI for the CHATIDEA framework was designed to leverage the visual layout of chats, like those of instant messaging platforms such as Telegram (fig. 28) or WhatsApp (fig. 29). This way the interface can provide a familiar environment to introduce new concepts to untrained users. In fact, by simulating a conversation, users are able to contextualize the various annotation steps in the general flow of a possible conversation scenario and better understand both the tasks at hand and the role of the data in the conversation.

4.4. Uniqueness of the Methodology

Similarly to other available frameworks for rapid prototyping of conversation agents, CHATIDEA is still based on intent matching and entity definition. Despite that it does not rely on user input for the intent definition. In fact, intents are mapped onto specific actions that can be performed on a database and are of four types: *start intents*, *history intents*, *find intents*, and, lastly, *filter intents*. For the scope of the annotation procedure, only the last two, find and filter intents, are of interest. Find intents are sentences that ask for certain information like "show me Professor Mario Rossi". Filter intents correspond to more complex queries to the database that require some kind of operations on the data. With this intent, the user queries for data that have specific characteristics. An example could be "Show me professors who teach bioengineering" where "who teach bioengineering" is the characteristic by which the results will be filtered. This peculiarity was leveraged in the design of the user interface for the annotation process and was what made the user of a conversational paradigm possible. In fact, since the type of intent is already known, partially written intents could be created so that they would act as text prompts of an hypothetical interaction between users and chatbot, leaving users only the task of entity generation.

fig. 28
Telegram UI

The entity generation essentially coincides with the database schema annotation phase. As the data is the subject of interaction between chatbot and end-user, entity definition resolves itself in the correct parametrization of the request in terms of tables and attributes. So, if the chatbot designer has an intent such as "are apples available?", what would happen in a framework like Dialogflow or Amazon Lex would be that the designer must define an entity for the food items available. This action essentially creates a table of food items. In the case of the CHATIDEA framework, the table with the food items already exists so the designer does not have to create it from scratch. What is left to do, instead, is linking the table to a conversational element, the word *apple* in the example, so that when the users ask for apples the chatbot knows in which table to search for the value *apples*. This linking operation is achieved through the annotation process.

What is above here described is unique to CHATIDEA. Currently, in fact, there is no tool that allows database to be parsed and used directly to generate intents and entities. In addition to that, the fact that intents are mapped upon SQL commands, granted the unique opportunity to design conversational paradigm based on a single dialogue template reusable for any database.

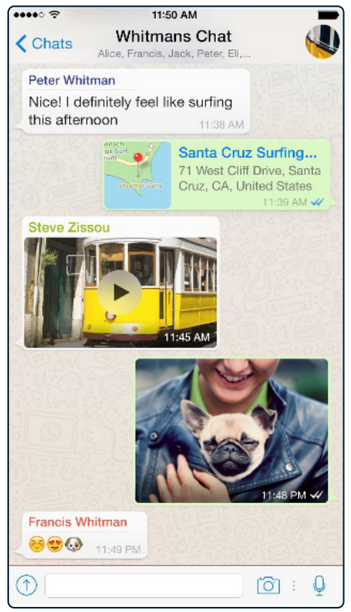
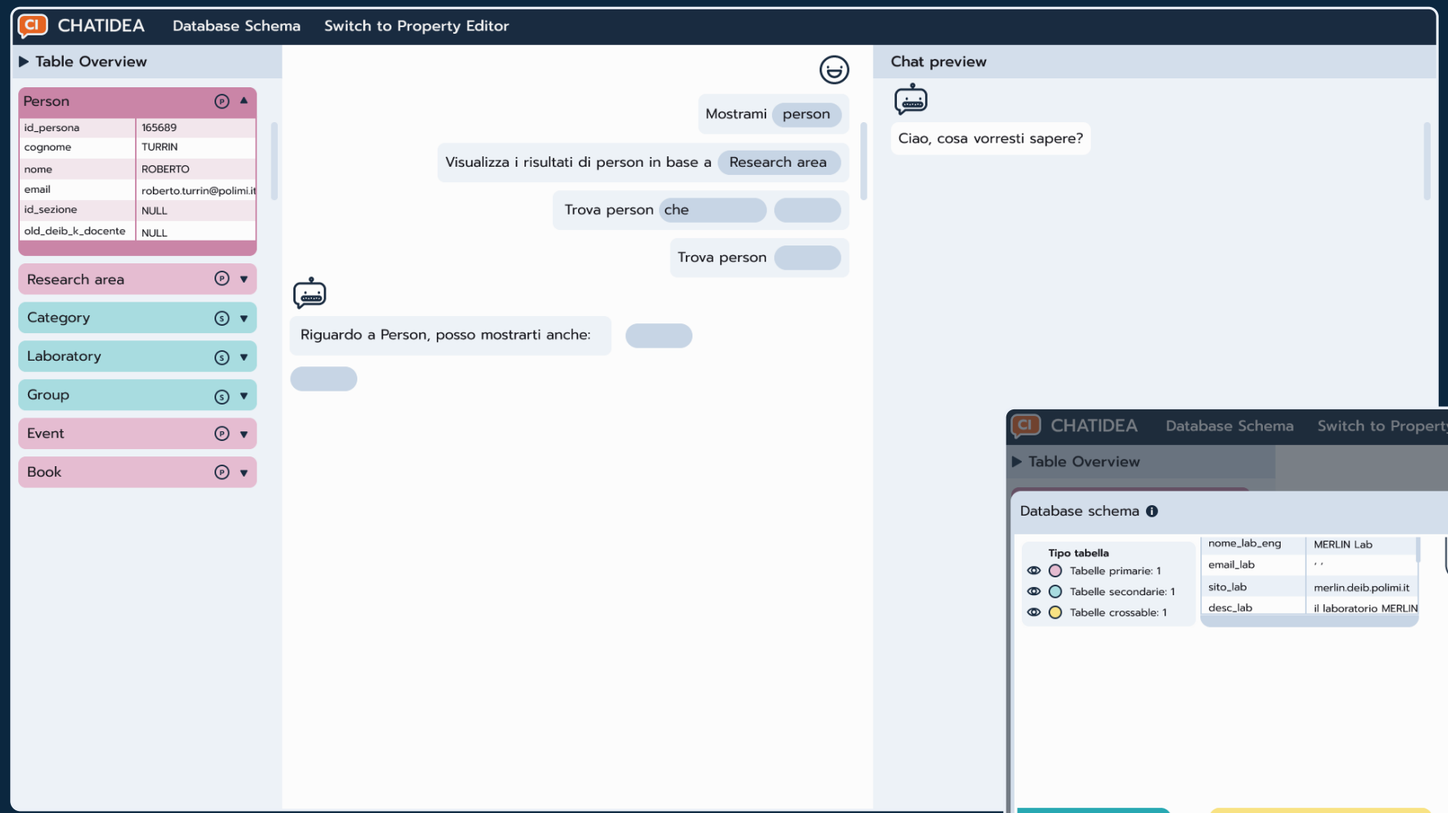


fig. 29
Whatsapp UI

5. Design of the interface

CHATIDEA



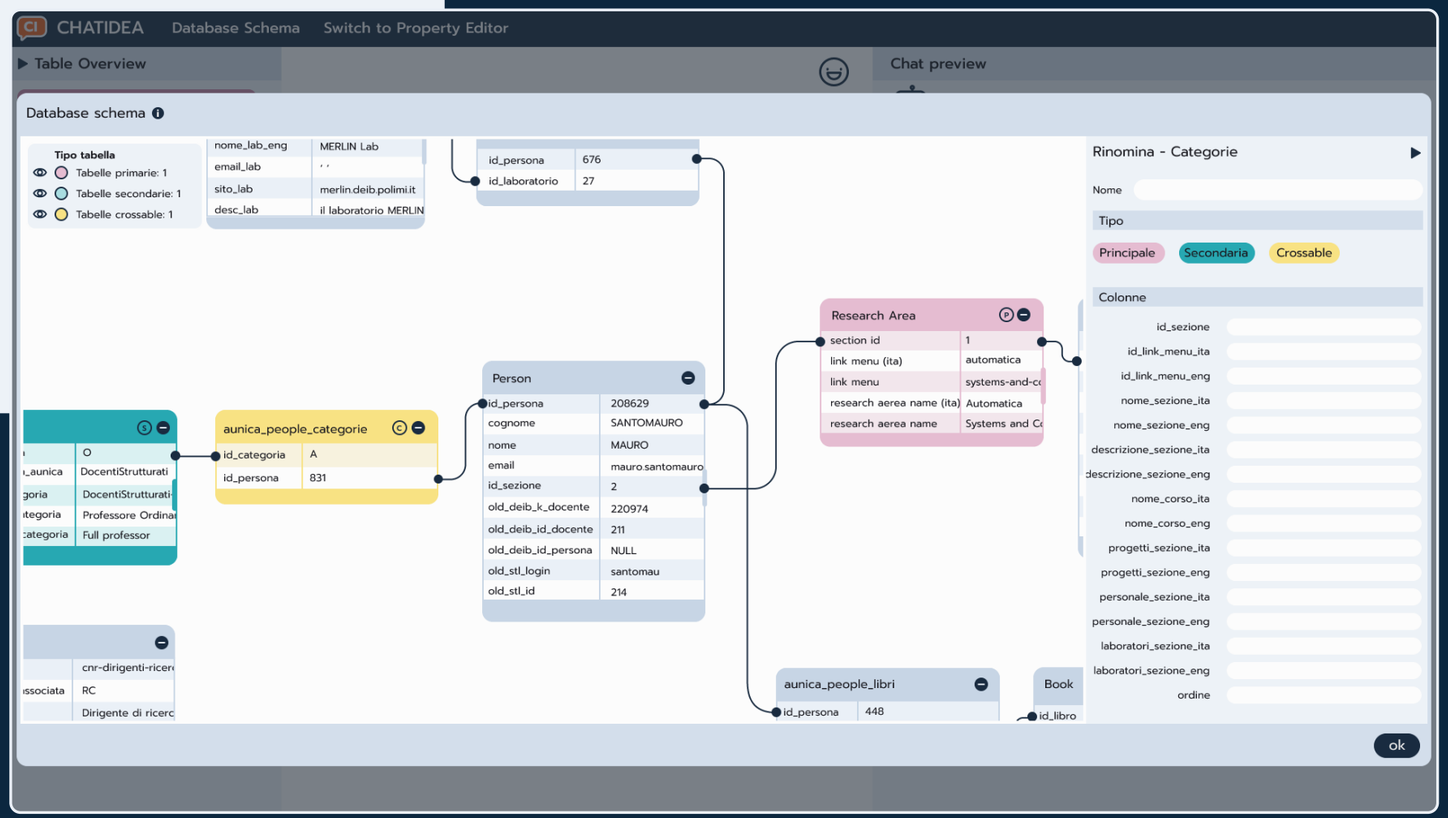


fig. 30
In the pages before: two windows of CHATIDEA GUI

5.1. Summary

Designing an EUD tool was an all-round project that encompassed both User Experience Design and User Interface design. This chapter starts by stating concept and target at the base of the design process. Then it presents the visual identity, delving into the design of the logo, and the various visual components, from the icon system, to the colour choices, to data visualization.

5.2. Concept

The CHATIDEA graphical interface is an End-User Development tool for the generation of the database schema annotation files, which are the starting point of the framework for the development of chatbots for data exploration. The interface exploits a conversational paradigm to guide the chatbot designer through the process by the means of text prompts modelled on a hypothetical conversation that the end-user may have with the chatbot. The text prompts are skeletons of training phrases that the designer completes with keywords and data taken from the database. This way the designers are able to create the mapping between data elements and conversational elements, within the context of a sample conversation that helps in understanding the possible patterns of navigation in the database.

Furthermore, as the designers complete all the prompts, the tool will automatically generate the database schema annotation files without the need for them to write any code or know in detail all the concepts necessary for defining dialogs in conversational agents. Thanks to this, the designers are helped in two fundamental ways: firstly, they are provided with a set of examples that suggest the possible interactions with the data, making the database navigation easier. Secondly, to use the framework efficiently, designers do not have to know well the concepts that the framework uses for generating the dialogue. Most of the annotation categories are hidden from the designers, that simply complete the prompts given to them.

5.3. Target

As chatbots start to be employed in more and more sectors, from retail, to healthcare, to customer care, the pool of users also diversifies. In fact, not only programmer users can be interested in the development of a chatbot, but also employees belonging to the marketing department or CRM department. As the output for this framework is a chatbot for data exploration, it can be particularly useful as an intranet resource to help employees navigate their firm databases. The interface here presented aims at making the framework more accessible, aiding the chatbot designer in completing the database schema annotation procedures. Taking into account the diversity of scenarios and user cases, it was designed both for programmer users that are not expert with the framework, and for non-programmers that are not familiar either with the JSON format or the framework. One important consideration to make is that to be proficient with it, the user will still need to have basic knowledge of relational databases visualization.

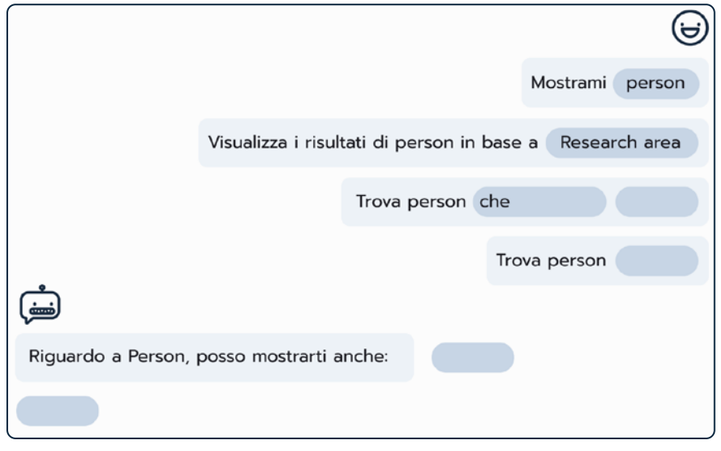


fig. 31
The layout of the conversation editor. The placing of the text bubbles references instant messaging platforms

5.4. Visual Identity

The interface proposed in this work contains a lot of textual information, due to the nature of the tasks that compose the database schema annotation procedure. To ease the cognitive load posed on the user, a system of colours and icons was designed to encode concepts and actions that did not require text input, while to visualize the annotations a conversational paradigm was employed.

As we have seen, the interaction paradigm chosen for this tool is a conversational one. Thus, the visual identity of the interface was modelled after that of instant messaging platforms, in order to create for the user an environment that could feel as familiar as possible. The main inspiration for the layout and visual elements was drawn from instant messaging applications such as Telegram and WhatsApp. Text bubbles-like graphic elements were designed to showcase the editable intents. To differentiate intents that regard user utterances from intents that refer to chatbot messages, the position of the text bubbles mirrors the one used in messaging platforms. In addition to that, the bubbles are preceded by an icon that identifies the user and one for the chatbot.

5.4.1. Colour palette

The interface uses colours as signifiers to visually represent different abstract concepts. The annotation procedure is, in fact, composed of many different steps that need to be graphically conceptualized to make them more accessible to unexperienced user base. One of the most important concepts is the *type annotation*. As we have seen, three different types can be assigned to tables, *primary, secondary,* and *crossable*. These three values were encoded with colours to increase the readability of the database and give a permanent visual cue that could help users in recognizing at a glance the different types of tables.

For the background elements a neutral blue palette was chosen to set off the colours of important elements like the tables and annotations. For buttons a dark blue tint was chosen to make them contrast with the background, while maintaining a cohesive look. Lastly, for the annotations two colours were chosen: a light blue shade already in use for the tables that do not have an assigned type was used to highlight the annotations that have yet to be made. Instead, the orange colour was chosen to mark the completed annotations.

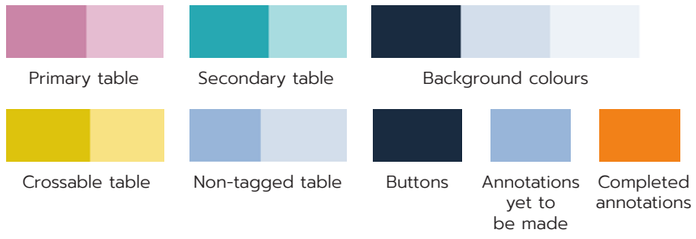


fig. 32
The colour palette for background, buttons and annotations

5.4.2. Icon system

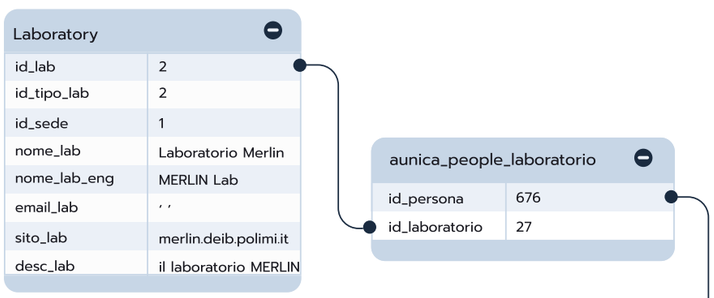
Icons provide pragmatic metaphors that help users in concretely represent abstract components (Marcus, 1998). Using familiar concepts in an unfamiliar environment can increase ease of learning and memorization. On the other hand, they can be dysfunctional if they cannot convey their function clearly. This issue was very much relevant in the design of the icons used for this interface. In fact, as the tasks and concepts are mostly abstract, it was not always possible to identify a visual metaphor to represent them. In some cases, colours where used, as we have seen with the *type annotation*. However, a set of icon was designed for buttons functions and to identify the user and the chatbot in the Conversation Editor and in the Chatbot Preview.

In particular, for better accessibility, for the *type annotation* the colours were paired with a symbol made from the *type* initial. This way, users could also have a symbolic representation of the type to help them in identifying them.
For the *conversational type* annotation, a set of icons representing each one possible data type was designed, to give users a visual queue when setting this particular annotation.
A pair of eye icons were designed to implement an hiding functions to give user control on how many tables they can visualize at the same time. Then a link icon was designed to highlight foreign keys. This icons makes such attributes recognizable and grants access to a specific action that we will see in the following chapter. Lastly a save icon was designed to validate user input.

fig. 33
Icons Designed for the chatidea framework

5.4.3. Data visualization

Since the framework is based on relational databases, the type of diagram used to visualize the data is based on entity relationship diagrams, ERD; the type of diagram most commonly used to represent such databases. Compared with ERDs, the diagram used in this work has a less technical visual syntax to make the visualization more understandable to an inexperienced user base. In particular the syntax pertaining the nodes. Nodes can be represented using different kinds of notations. In figure 35 we can see an example of crow's foot notation, also known as Martin notation (Everest, 1976). The crow-foot notation, where the ends of the nodes differ based on the kind of relationship they represent was relaxed avoiding to use the symbolism tied to type of relationship existing between two tables in favour of a single type of node with circular ends. In addition to that, the tables now show also the first value of each column, to aid the user in understanding the content of the table, as the column's name may not be representative enough.



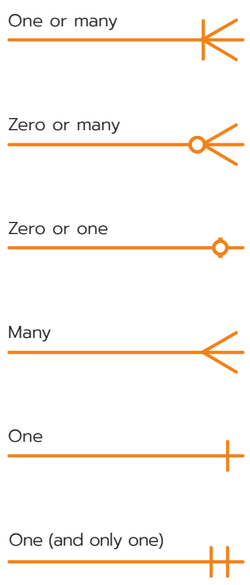


fig. 34, **fig. 35**
On the left, an example of the relational diagram employed in the interface Above, examples of the crow foot notation

5.4.4. Logo

The name CHATIDEA, is the acronym of CHATbot for Interactive Data ExplorAtion. Starting from this we set out to create a new logo for the framework that could convey the principal characteristics of the framework: the conversational nature of the interaction paradigm used in the tool and in the exploration of structured databases. Thus, the logo (fig. 36) of CHATIDEA was designed to communicate visually the concept of chatting and conversation. For this reason a text bubble was chosen to represent this concept in light of its immediate recognizability and memorability. In addition to that, since annotations are the most important and characterizing concept of the framework, the orange tint used for the annotations was reused in the logo. Inside the text bubble are placed two letters from the name, *CI,* so to maintain an hint of the original name even when the icon is used alone.

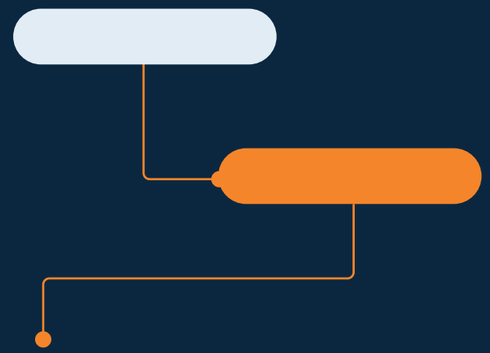
The typeface used (fig. 37) is the same as the one used for the whole tool, Prompt by Cadson Demak. A sans serif font with airy negative space that works well in small size and is suitable for to be used on the web, but also in print.

fig. 36
CHATIDEA logo

fig. 37
Typeface speciment

Prompt

Sphinx of black quartz, judge my vow
Sphinx of black quartz, judge my vow



6. Information architecture

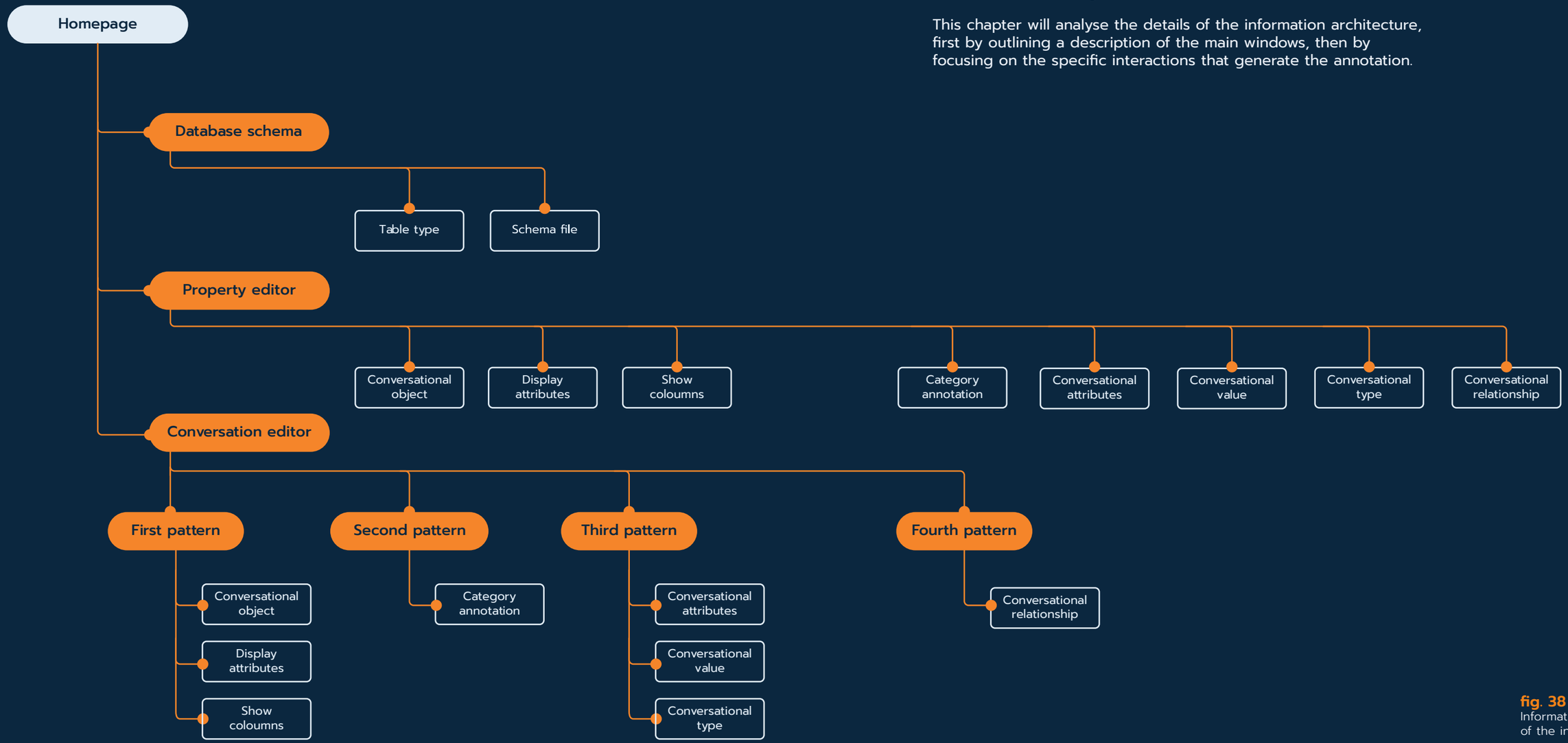


fig. 38
Information architecture of the interface

6.1. Summary

This chapter will analyse the details of the information architecture, first by outlining a description of the main windows, then by focusing on the specific interactions that generate the annotation.

6.2. Home

The tool opens with a homepage (fig. 39) that gives the chatbot designer the option of creating a new annotation project or opening an already existing one. If the designer chooses to create a new one, she or he has to give it a name and import the relational database schema using the button next to the call to action.

6.3. Database Schema Screen

After having imported the schema of the relational database that needs to be annotated, the designer is presented with the *Database Schema* Screen shown in figure 40. This screen is devoted to the management of the database and the definition of the *Schema file*. Here the chatbot designer is presented with

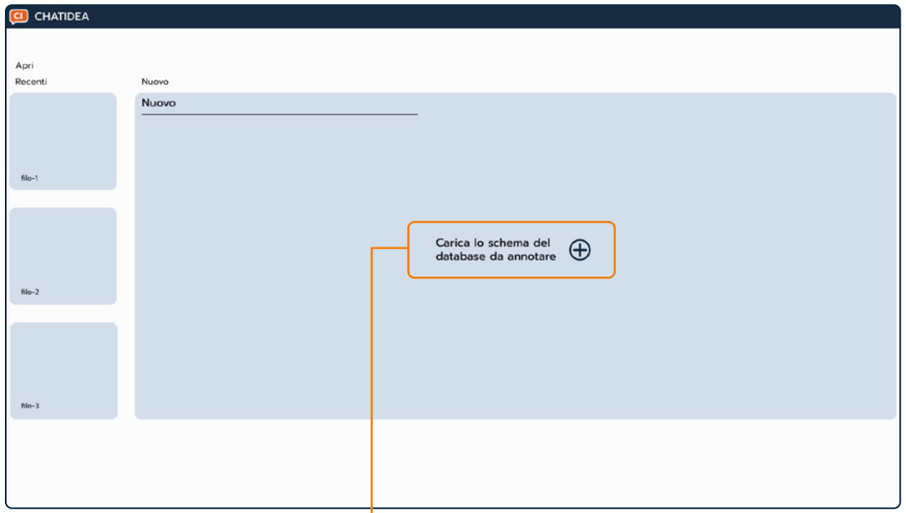
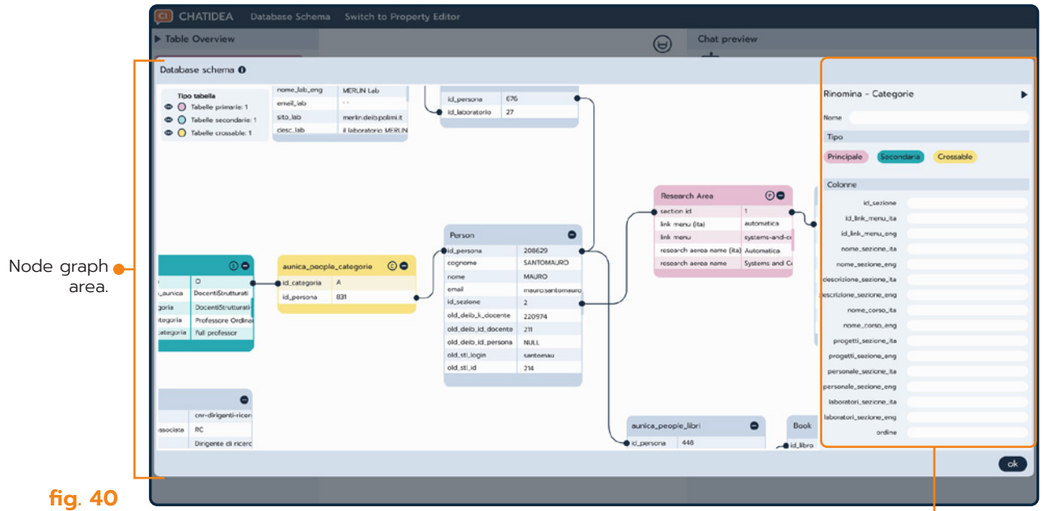


fig. 39
Homepage

Users can add the database schema from here

a node-graph representation of the database, that highlights the relational structure of the data. On the right, there is a sidebar containing all the editable elements and properties of the database. Through the node-graph the designer can skim the database, choosing what tables will be needed in the conversation with the end-user and what will not. A database, in



Node graph area.

fig. 40
Database Schema

Sidebar editor

fact, can contain tables that have no use in a conversation with an end user, but that still have a reason to be in the database, so the designer can omit them using the minus button. The table will not be deleted from the database, but just omitted from the annotation process. Next, the designer can edit the relations between tables and draw new nodes by a dragging interaction. This way the designer can add missing relationship meaningful for the conversation with the end user.
Lastly, on the top left of the node-graph viewport, there is a quick interactive legend that allows the user to toggle the visibility of tables that have been assigned a type. With this feature the designer can easily keep track of the *Type* annotation, thus avoiding having too many tables of one type or another. Moreover, it improves readability of the database as less tables can be in view.

In the sidebar editor, instead, the designer can edit the tables and attributes names and decide their role in the schema, meaning their type: primary, secondary, or crossable. This last annotation, actually, is part of the *concept file*, but was included

in this editor because it is propaedeutic for following annotation concepts, like the category annotation. Furthermore, it gives important hierarchical information that can help the user in the understanding how to structure the conversation with the end user. Moreover, table marked as crossable, will be excluded from the following annotation steps, so it was fundamental that the table *type* annotation was done in the early stages of the annotation process. Tables have to be marked as crossable instead of simply being omitted with the minus button because they are needed for the generation of following annotations like *conversational relationships* and *conversational attributes*, as they are links between tables in a many-to-many relation.

6.4. Concept File Annotations Screen

After having completed the database management tasks, the designer accesses the screen shown in figure 38, devoted to the creation of all the other annotation concepts contained both in the *Display file* and *Concept file*.

The screen is divided in three parts. On the left it can be found the *Table Overview*, which is a sidebar listing all the tables tagged as primary and secondary. The user can expand them to easily check the content of each of them. The *Table Overview* can also be expanded to get a node-graph visualization, so that also the various relations between tables are on display (fig. 41). At the centre of the page, there is the main content, where the designer performs the different annotation tasks. To define the

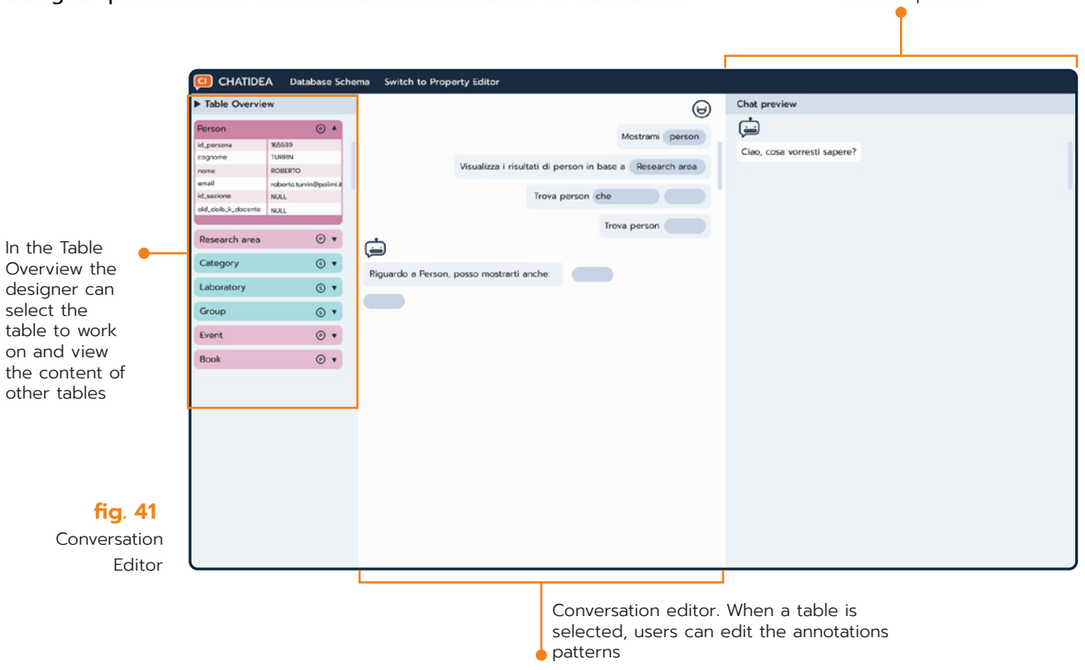


fig. 41
Conversation Editor

annotations, the designer can choose to use the *conversation editor* or the *property editor*.
Like already said in the previous chapter, the *conversation editor* exploits visual metaphors used in conversational user interfaces to show the user the different editable concepts. The annotations are presented to the designer as editable messages sent by two different senders: the chatbot on the left and the end user on the right of the main content. Much like profiles pictures, the annotations are introduced by two icons, one for the end user and another one representing the chatbot.

The *property editor*, shown in figure 42, is an alternative editor that can be used to generate the annotations. It was designed for more advanced users, that know very well the framework. One of the principles of End-User Development, indeed, refers to the provision of different abstraction levels (Green & Petre, 1996), to ensure a gentle slope of complexity (Lieberman et al., 2006), thus accommodating different user skills and attitudes and also varying usage contexts. In the editor, the concepts are divided in accordion sections that can be expanded or contracted as needed. With this editor the designer can readily access all the annotation fields either to perform the whole procedure or to make quick changes.

The same structure seen in the conversation editor is also kept for the *chat preview* on the right. This area, simulating a messaging platform, is used to get a real time preview of the result of the annotations. Thanks to it the designer can have live feedback on the process and a clearer understanding of the task at hand.

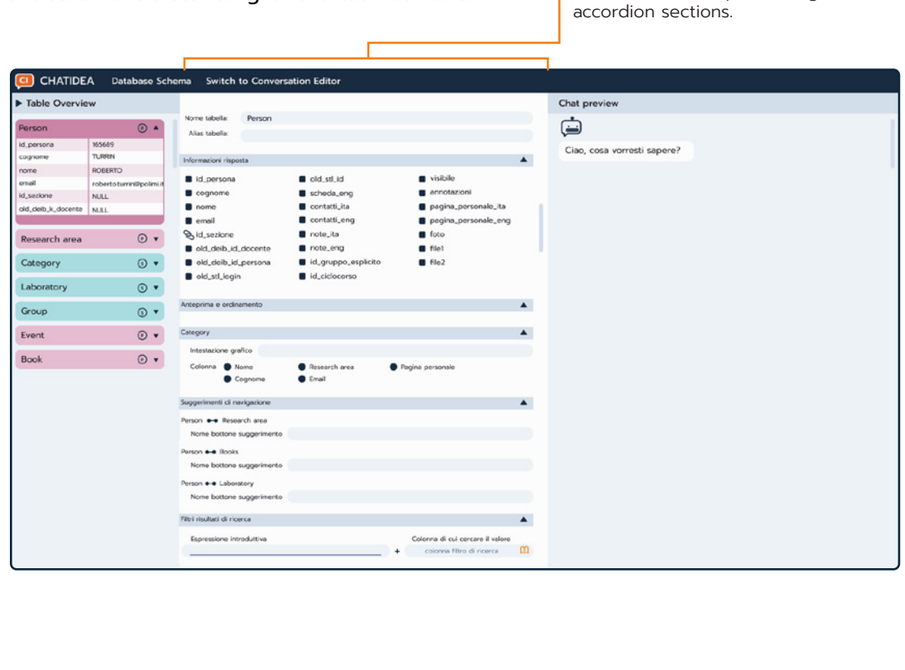


fig. 42
Property Editor

6.5. Concept File Generation

To begin the concept file annotations, the designer must first select one table from the sidebar on the left of the screen. By doing so, a list of text prompts presented as messages will appear. Each prompt has a highlighted part that can be clicked to open a dropdown containing all the editable fields. After having edited all the prompts, the user can select another table and continue the annotation process. Each text prompt covers a different concept of the annotation process. We will now see how the user interacts with each of them and what concepts are annotated by doing so.

6.5.1. First pattern: conversational object annotation, display attributes, and order by

Clicking on the first pattern highlighted element opens the drop-down menu (fig. 43) covering the conversational object annotation, the display attributes, and the order-by annotation. They have been grouped together because these three annotations are part of the same intent: finding data. First, the designer writes aliases, i.e., alternative names by which the end users may refer to the table they are looking for.
The first word written in the list of aliases is, by default, used as the conversational object in the actual JSON, while the following ones serves as aliases. After this step, the designer checks what attributes she/he wants to use to answer the query. After being checked, each attribute has to be renamed so that it is more understandable for the end user. Attributes marked by a link pictogram are foreign keys. When the designers select them, they can access attributes from the referenced table to show instead. Foreign keys in the origin table, indeed, are just identifiers that could not make sense for the user, while the data in the target table would be more meaningful.

Lastly the designer checks what attributes will be used to summarize the answer when the chatbot returns a long list.
By design of the annotations, the attributes checked will also be used to order the results in the list.

6.5.2. Second pattern: category annotation

The category annotation (fig. 44) is done in the second pattern. Here the chatbot designer chooses by which attribute the content of a table must be visualized. The system can suggest what attribute could be the best fit, but the designer is free to choose one or more attributes. The result will be a pie chart that

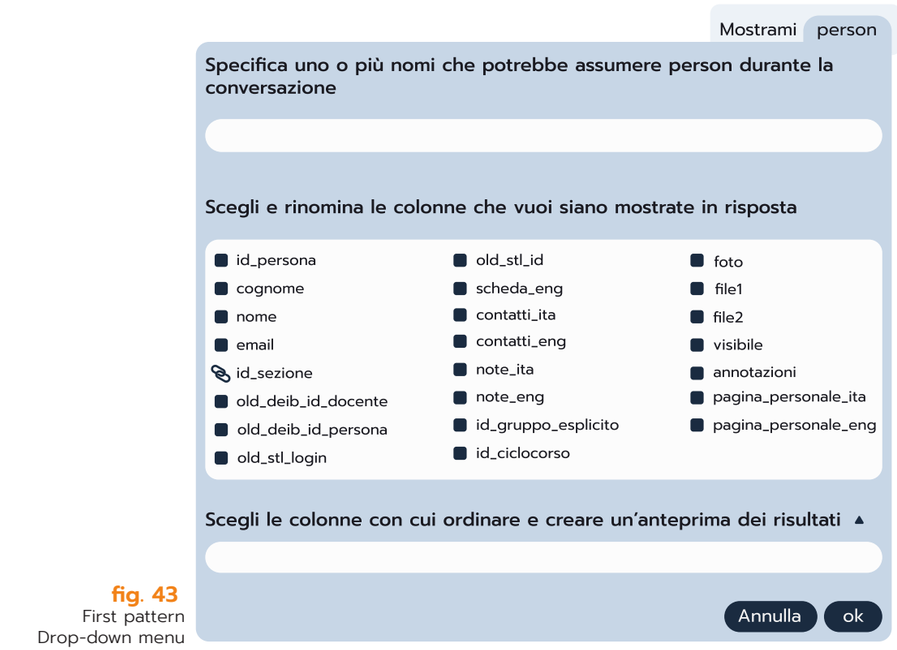


fig. 43
First pattern
Drop-down menu

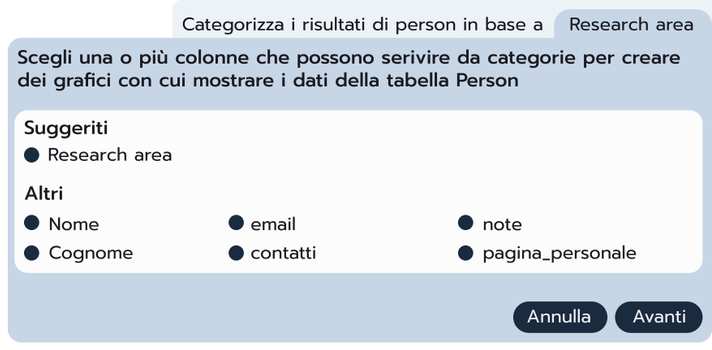


fig. 44
Second pattern
drop-down menu

quickly summarizes the data. After having picked one attribute, the designer can type a new name for the attribute to generate the legend that will accompany the visualization (fig. 45).

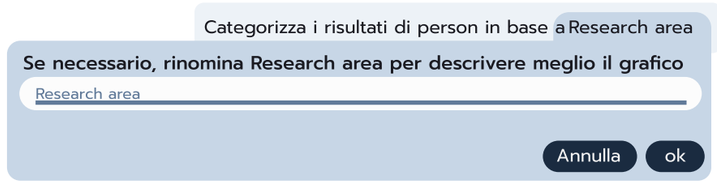


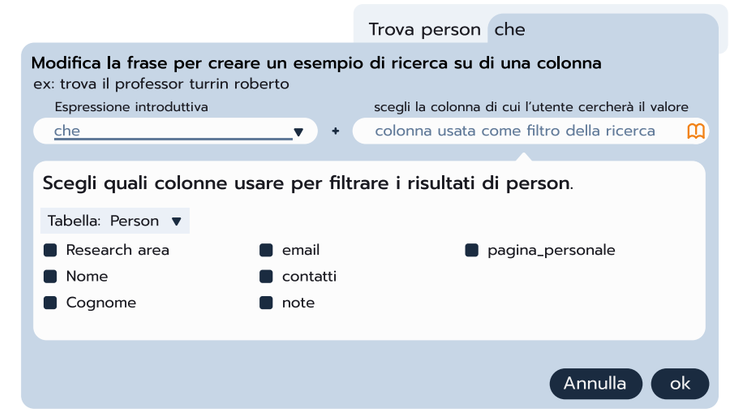
fig. 45
Text field to generate the visualization legend

6.5.3. Third pattern: conversational attributes

This annotation covers the filter intent and is more complex than the ones before. The designer is called to write a keyword in the text field on the left. The keyword can be a phrase or just a word, that introduces the data that acts as a filter, as in the conversational value (i.e.,each value of a chosen attribute). Then, the designer has to pick the attribute where the chatbot will look into to find the conversational value. To do this, first, the designer clicks on the right field. After that, the chatbot designer chooses from a drop-down menu the table where the data is stored, then the designer checks the attribute from the multiple choice.
To complete the annotation, the chatbot designer must specify the conversational type by checking the icon corresponding to the correct data type situated next to the field reserved for the conversational value. A second instance of this pattern is reserved for defining queries that do not have a keyword introducing the conversational value.

6.5.4. Fourth pattern: conversational relationships

Conversational relationships, as we have seen, in the final interaction between end user and chatbot are conveyed through buttons that are used to give the end user suggestions on how to continue navigating the database. For this annotation, the designer first chooses what table the button leads to; then, she or he types a new and more descriptive name that acts as a navigation tip.



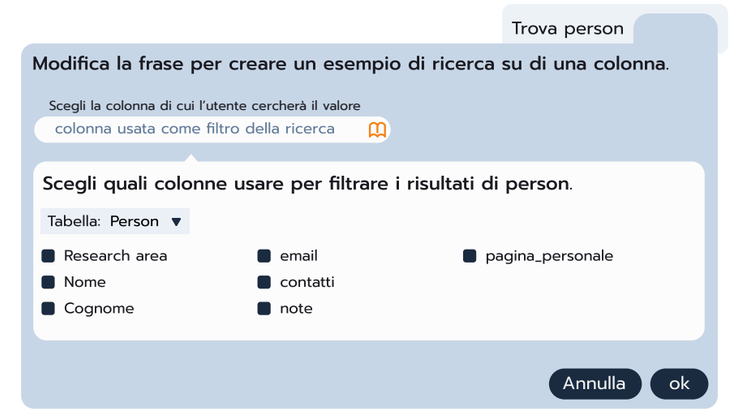


fig. 46, fig. 47
Above, the first pattern for conversational attribute structured as so:
Find <keyword> <conversational value>

Below, the second pattern. This pattern is structured as so:
Find <conversational value>

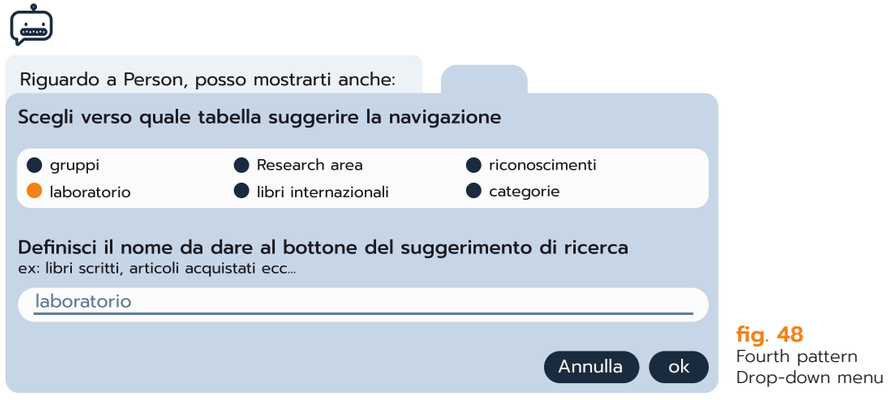


fig. 48
Fourth pattern
Drop-down menu

7. User-based Evaluation

7.1. Summary

This chapter will discuss the results of the three user evaluations conducted during the development of the interface. For each evaluation study the we will see how the test was conducted, what where the insights gained and the main takeaways.

7.2. Preliminary Evaluation

To design and test the user interface a prototype was developed by using the rapid prototyping software Figma. After having completed a first iteration of the UI prototype, a group interview was set up with the four CHATIDEA framework developers. The developers are four former students from Politecnico di Milano and University of Bari that recently graduated in Computer Science and Engineering. One of them developed the first version of the framework presented in (Castaldo, 2019), while the other three worked on further extensions and improvements to the framework presented in the thesis(Ferreri & Notari, 2020).

The interview revolved around a guided demonstration of the prototype. It was intended to get feedback from a pool of users that had already performed the annotation task without the aid of a visual tool. This way we could assess the validity of the design choices and interaction paradigm, and in particular its correctness with respect to the origin methodology. Due to the restriction imposed by COVID-19, the interview was held through Skype and recorded with the participants approval.
The prototype was broadcasted from my computer so that the participants could follow the demonstration of the interaction with the prototype. The interview was structured in two parts.
During the first part of the interviews, the participants were presented with the prototype. Due to the limited freedom of interaction given by the prototype, I acted as a guide and gave a demonstration of how to perform each annotation task. After this initial phase, during which I went through the prototype from start to end, there was a discussion session during which the developers could make their comments regarding

the interface and we could gather insights. Starting from the beginning of the demonstration, feedback was recorded for each step of the interaction. All the participants considered the conversational paradigm clear and effective, while specific issues were found regarding some tasks and the introductory message that explains them.

The most important problem highlighted was that the participants did not understand that to get the text prompts they had to first select a table. In that iteration of the interface, in fact, the first table of the database was already selected.
So, the user was immediately presented with a set of prompts for that table, but not for the rest of the database.
Another significant problem was encountered with the table types: according to the presentation of the prototype given at the beginning, any table could be tagged with the type crossable, while, on the contrary, participants noted that only tables with at least two keys can be tagged as such.
A further issue was found with the editing of relations. As initially defined, new relations could be drawn by dragging a handle situated on the table header towards another table header, while, technically, the linking is between keys.

Lastly, in two different occasions the explanatory message introducing each task was found too vague. In particular, the message explaining the category annotation task was not clear enough in explaining the outcome of the task. Another significant issue was found in the show columns annotation explanatory text. The text explained only one of the two functions of the annotation, explaining only that the annotation served to order the results, but omitting that it also determined which attributes would be shown when returning a list of results.

7.2.1. Takeaways

Improved copywriting
After the evaluation, the message introducing the *category* annotation was changed to include the table name in the task description. Other tooltips were reworded to were reworded to better express the tasks and clear ambiguities, In particular the tooltip that welcomes the users during the first user was modified to include more propaedeutic information to start interacting with the tool.

Warning message for tables wrongly tagged as crossable
To prevent users from tagging tables with only one key as crossable, a warning message was added. So when users wrongly check the crossable type for tables with only one key, the warning will appear (fig 49).

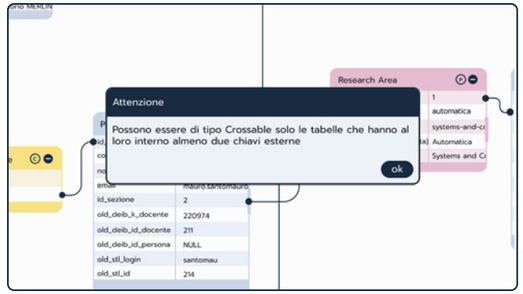


fig. 49
Warning for wrogly tagged crossable tables

Editing relations
Following the feedback of the framework developers, the relations drawing was changed (fig. 50) from table header to table header to table key to table key. This solution requires the user to have some knowledge of relational databases, but is less ambiguous when tables have multiple keys referring to as many tables.

Redesign the start of first-opening state of the conversation editor
Following the remarks made during the interview, the default state of the conversation editor was redesigned (fig. 51). Instead of presenting the user with an already chosen table and the respective text prompts, the user finds the editor empty except for two system messages that explain what the editor is for and how to start editing the annotations. This way the user is guided in choosing the first table and can learn how to use the database overview sidebar.

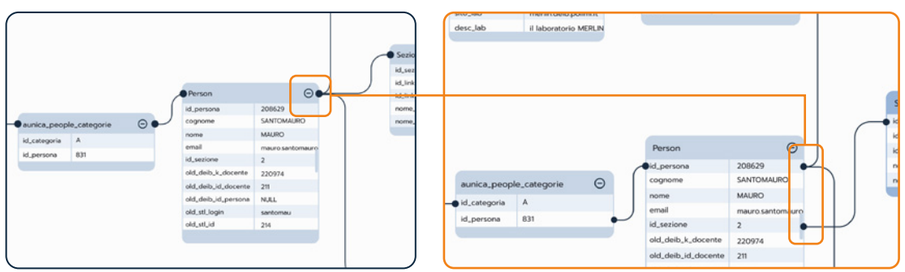


fig. 50
In this figure we can see how the nodes' handles were moved from the table's header to the primary and foreign keys

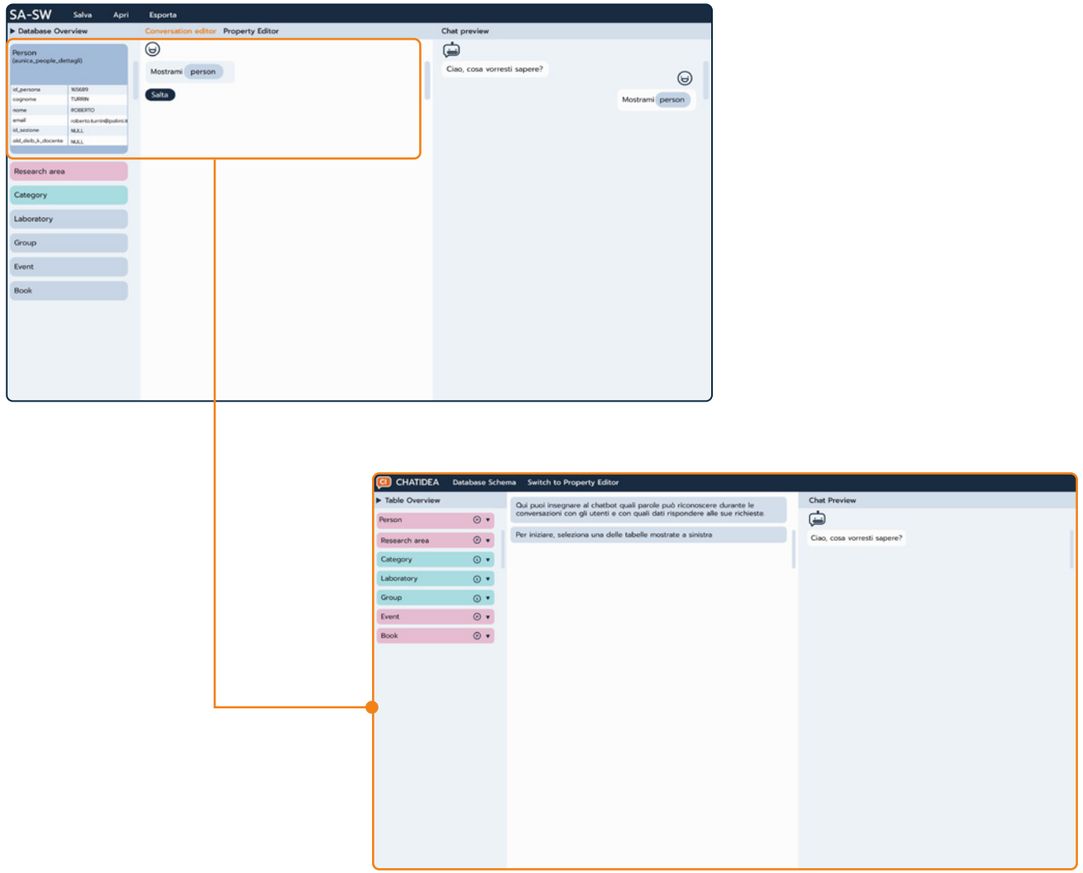


fig. 51
The starting state of the conversation editor was changed from having a table already selected, to having none. Moreover, a new help tip was added to guide the initial interaction

7.3. Second Evaluation Study

After this first group evaluation, a second one was held with Emanuele Pucci, CEO and cofounder of Awhy[13], a company specialized in the development of chatbots for customer care and support. The interview was held the same way as the previous one. I presented the prototype by broadcasting it though Google Meet and explaining all the points of interactions. After this initial phase, we moved onto the question session. This time, the participant did not find significant issues with the interface itself and regarded the prototype he was shown as being clear and intuitive. He appreciated a lot the paradigm and also the opportunity it gives to define chatbots on top of structured databases. According to his experience, the need for this kind of chatbots is now emerging in companies. In particular, he noted that it would be especially useful for intranet purposes, to develop chatbots that could help employees in navigating the company databases. However, the participant expressed doubts regarding the scalability of the interface with extensive databases. He noted, in fact, that in a large database it may be difficult to keep track of all the relations between tables. This issue regarded both the graph visualization of the database specifically, and the whole annotation procedure.

7.3.1. Takeaways

This evaluation session was focused especially on the acceptability of CHATIDEA and its visual paradigm. It allowed us to find out whether a framework to define chatbot on top of a structured database, like CHATIDEA, would be of interest in the industry. The result was positive, confirming that there is an interest in the industry, in particular for enhancing the access to structured content such as the one generally available in companies' intranets.

The most significant take-away for the UI design regarded the scalability of the provided visualization of the database schema in the first phase of design, when the role of the different tables has to be marked up. The concerns that came out were related to the difficulties of : 1) displaying the schema for large databases, which could have a high number of tables (sometimes more than fifty); 2) configuring all the queries for each single table. Due to time constraints this issue could not be addressed, but future work could focus on making the interface more scalable. For example, a progressive visualization of the database schema could be devised, which could start from "central" tables, i.e., those tables with the higher number of outgoing and ingoing relationships, and then progressively explore the other tables. Also, to reduce the workload on users,

13 https://www.Awhy.it

some tasks required for the configuration of the user queries, like type annotation and category annotation, could be partially automated by analyzing a-priori the database schema and the data stored within the tables.

7.4. Third Evaluation Study

A third evaluation study was carried out to further validate the interaction paradigm and to understand how effectively users could interact with the tool. For this study, a running web front-end application was developed to provide the participants with a realistic interaction. The front-end was not connected to the engine of the framework in charge of generating the chatbot code, but it was fully working and supported the users in the different steps required to annotate the database and configure the conversation.

The front end was configured to support the definition of a chatbot on top of a sample database derived from the one serving the DEIB web site[14]. As reported in Figure 52, the adopted database includes 8 tables, a number that was not critical for the visualization of the database schema and that at the same time was reasonable to let the users make a selection of tables and their roles. The tables including central contents were considered (e.g., among the others, the ones related to the DEIB faculty, the awards, the published books, the research projects), which could provide the users with the opportunity to define multiple threads of conversation. This study was sustained by three groups of users, one group of programmers with experience in chatbot development and that have a thorough knowledge of relational databases. The second group was composed of users that match the target of this tool, namely non-programmers and people that do not have a solid knowledge of relational databases. The reason for the involvement of the first group was to gain insights from a pool of users with experience in chatbot development and thus know the current practices and can better identify possible missing functionalities.

7.4.1. Participants

The involved users were 12, ten males and two females. Their mean age was 24.5 years. The participants have been classified by their skill level in three skill level groups: programmers expert in chatbot development, programmers not expert in chatbot development and lay users. The first group was composed by five programmers from the technology company Awhy specialized in chatbot development. The second group

14 http://deib.polimi.it

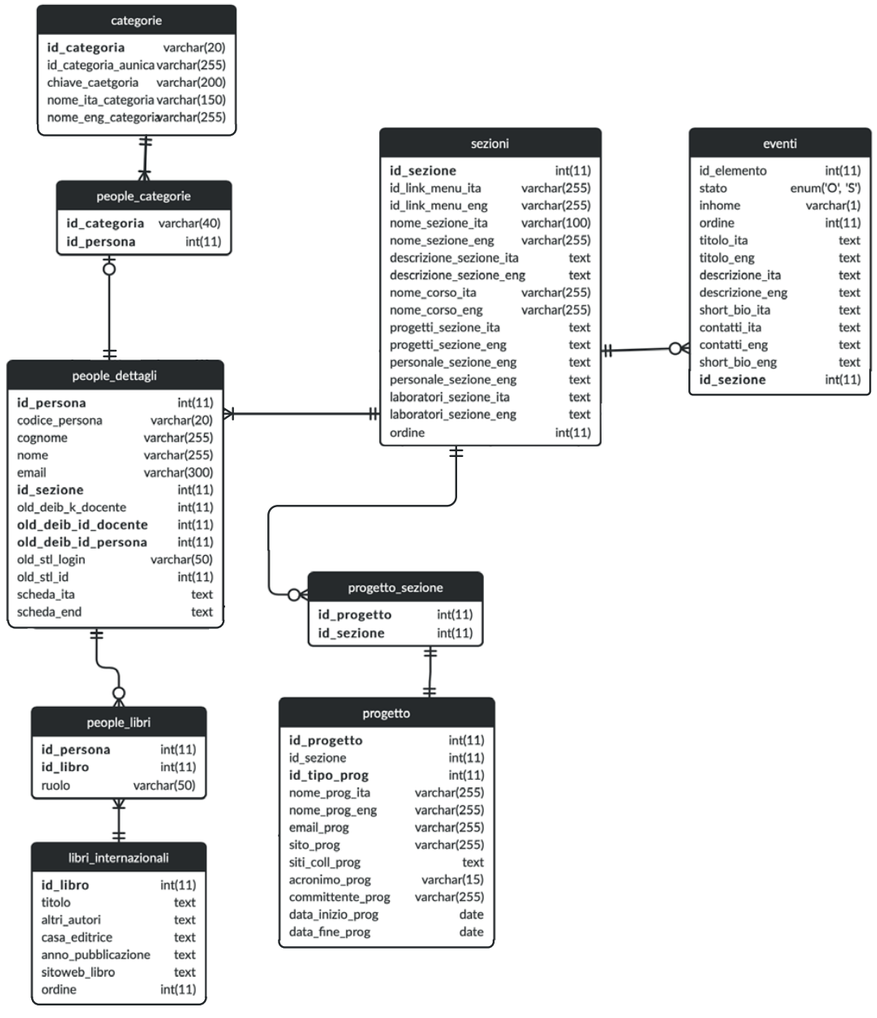
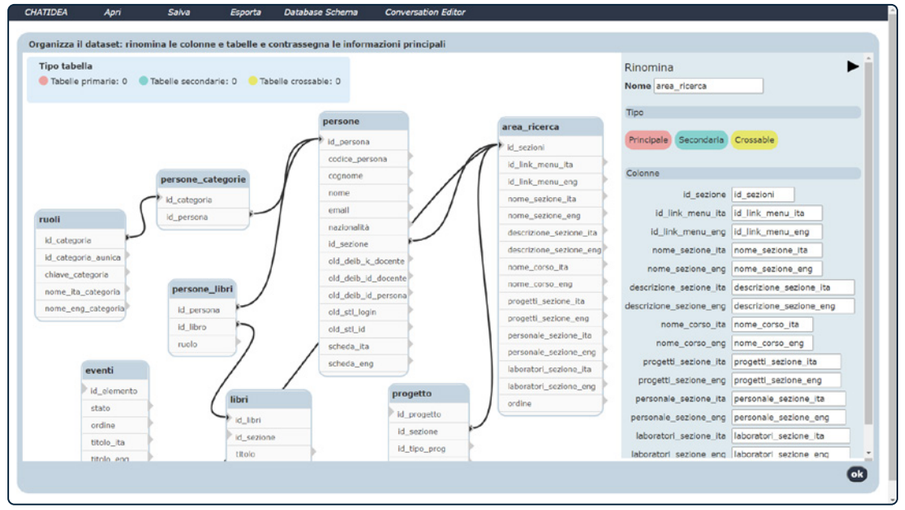


fig. 52
The schema of the database used for the user evaluation



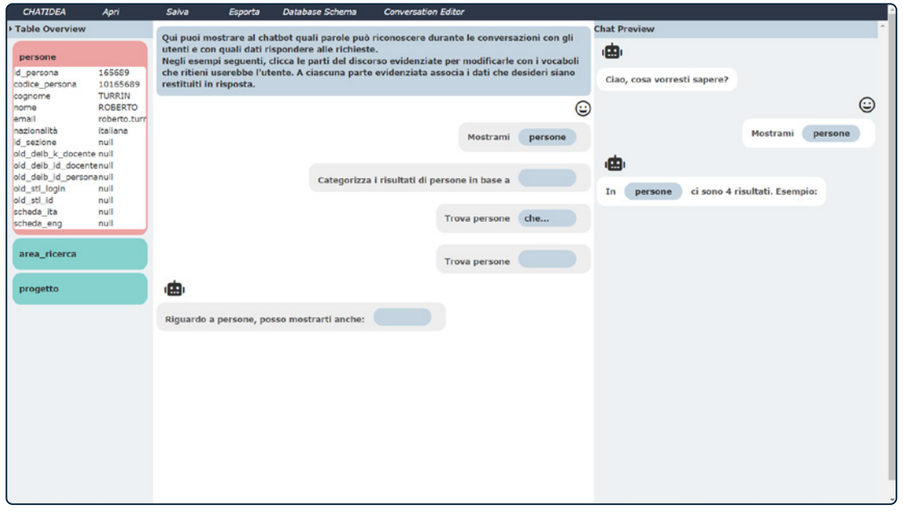


fig. 53, **fig. 54**
Windows from the front-end developed for the user test. Above the database schema, below the conversation editor.

was composed of students of Computer Science and Digital Communication from the University of Bari Aldo Moro.
The third group, instead, was composed of one game designer and three university students from different backgrounds: one bioinformatics, and two communication design students.

7.4.2. Procedure

We asked the users to participate in individual interviews during which we instructed asked them to perform 7 tasks for configuring a conversation on top of the sample database.
Due to the COVID-19 restrictions, the interviews were held online. Each participant was asked to connect to the online front end and perform the study tasks while sharing her/his screen. Each session was video-recorded. At the end, participants were asked to fill out a questionnaire. To prepare for the interviews, users were asked to read a document that provided a brief theoretical explanation of CHATIDEA together with a description of the scenario they would recreate during the interview.
The scenario gave some background to frame the objective of the activities they were going to perform. If the users did not have read the document, we gave an introductory speech that summarized the content of the document. During the study the users were invited to perform the task autonomously, but could ask for clarification if they needed. Each session was video-recorded with their consent. At the end of the interview session, each participant was asked to fill out a questionnaire. The first two tasks they performed, regarded the definition of the type annotation, while the remaining five tasks covered each of the other annotation concepts: conversational object, category, qualifier and relationship. The tasks were as follows:

Task 1
Define all the primary, secondary and crossable tables, depending on the dialogue you want to design. As primary table, you should select at least the table persone.

Task 2
If necessary, for the primary and secondary tables selected at the previous step and for their attributes you can change the names, to make them more understandable and adequate for the conversation in the final chatbot. When ready, you can click on the OK button to proceed to the next step of configuration.

Task 3
Select the table persone. Complete the structure of the utterance "Mostrami ___" ("Show me___").

Task 4
Complete the structure of the utterance "Categorizza i risultati di persone in base a ___" (Classify the results for persone on the basis of the attribute ___).

Task 5
Complete the structure of the utterance "Trova persone che___" (Find persone that____) ,by expressing a selection condition.

Task 6
Complete the structure of the utterance "Trova persone ___" (Find persone ____), by selecting one or more attributes for which the user can select some search keys to filter out some table instances.

Task 7
Complete the structure of the utterance "Riguardo a persone, posso mostrarti anche:___" (Regarding persone, I can show you also:___).

7.4.3. Data collection and analysis

Quantitative and qualitative data were collected for each participant. As quantitative data, the SUS score and NASA-TLX score were collected. SUS is a psychometric tool, used to assess perceived system usability (Brooke, 1995)(Lewis & Sauro, 2009). NASA-TLX was used, instead, to measure the perceived workload on the participants when using the system.(Hart & Staveland, 1988).

For the qualitative data, instead, comments made during the interviews by each participant were annotated and analysed, and systematised using inductive thematic analysis (Braun & Clarke, 2006). The data was organized into three groups corresponding to the skill levels groups used to classify the participants. The collected data were analysed, first, with an inductive approach to extrapolate relevant insights and generate codes, i.e., a system of labels that classifies the relevant features found in the data. The codes were, then, grouped into overarching themes.

7.4.4. Results

From the SUS questionnaire we recorded that the tool scored above the average (i.e.,a score of 68) (Sauro, 2011). The measured average SUS score was of $\bar{x}$ = 74.2, SD = 15.0; the score of the

System Usability was of $\bar{x}$ = 76.3, SD = 14.8; while the score for System Learnability was $\bar{x}$ = 68.8, SD = 17.3. The SUS score for the chatbot developers group was $\bar{x}$ = 74.4, SD = 15.2; for programmers $\bar{x}$ = 73.8, SD = 14.5; lastly for lay users $\bar{x}$ = 74.4, SD = 19.5. Regarding the workload measured through NASA-TLX, the value is measured on a scale from 0 to 100, where 0 equals to low effort, while 100 equals to excessive effort from the user. The average score recorded after this analysis was $\bar{x}$ = 38.1, SD = 13.3. The first group composed of chatbot developers scored $\bar{x}$ = 39.6, SD = 13.3; the second group of programmers not expert in chatbot development scored $\bar{x}$ = 39.6, SD = 6.0; while the third group of non-programmers scored $\bar{x}$ = 35.6, SD = 19.1. On average, the highest score, and therefore poorer performance, was recorded for the Mental Demand dimension ($\bar{x}$ = 54.2, ST= 17.3), followed by Effort ($\bar{x}$ = 44.2, ST= 20.7), Performance ($\bar{x}$ = 37.5, ST= 16.6), Temporal Demand ($\bar{x}$ = 31.7, ST= 24.8), and Physical Demand ($\bar{x}$ = 23.3, ST= 19.7).

Average by group	Mental Demand	Physical Demand	Temporal Demand	Performance	Effort	Frustration	Workload
Chatbot developers	52.5	32.5	27.5	40	40	45	39.6
Developers	57.5	17.5	40	40	45	37.5	39.6
Lay users	52.5	20	27.5	32.5	47.5	30	35
Total sample	54.2	23.3	31.7	37.5	44.2	37.5	38.1

table 3
Table summarizing the average NASA-TLX score by group and with respect to the total sample interviewed

Average by group	SUS Usability	Sus Learnability	SUS Score
Chatbot developers	77.3	71.9	74.4
Developers	75.0	68.8	73.8
Lay users	76.6	65.6	74.4
Total sample	76.3	68.8	74.2

table 4
Table summarizing the average SUS score by group and with respect to the total sample interviewed

The results of the thematic analysis highlighted common themes across the three skill level groups. The quotes here reported are a translation of the observation made by the participants in their native tongue during the interviews.

First theme: Conflict between system architecture and users' assumptions.
We observed that users assumptions about some tasks were in conflict with the system architecture. The majority of the users encountered difficulties completing the pattern "Show me <table name>. In this pattern users can choose first the attributes to give back in response about each record of the table; secondly, they can choose which attribute to use to generate a preview of the results and decide the order in which they are shown when many results are returned as a list. By design this second step is the same annotation in the JSON file. However, most of the participants did not understand the difference between selecting the attributes that would be used to answer the query, selecting those that would generate a preview of the results and decide the order in which they are shown when they are returned as a list. The tooltips of the operations were not clear to them and they mostly failed to perform this part of the task correctly.

> *"Am I choosing how to order the attributes shown in a single response, or am I choosing the columns to generate and order the preview?"*

Another assumption was that the attributes had to be renamed for conversational use in the database schema window.
So they were confused when they were asked to rename them when completing the first pattern, where they had to choose the attributes to show in response to a query.

> *"Will these names be used by the user while chatting?"*

Second theme: Knowledge of the domain of the database and of basic concepts of relational databases is a must
Users across all groups stated that knowing beforehand the database domain would have helped them in understanding better how to complete the patterns and how to choose attributes and relationships.

> *"It's difficult to use the interface without knowing the database"*

> *"I do not know what this attributes mean so it is more*

difficult to decide which should be chosen"

Lastly, the group of non-programmers stated that the tool would be more suited for people that have at least a basic understanding of relational databases, as they encountered some difficulties tasks that required basis knowledge of relational databases concepts: the help tip about crossable tables provided by the system was not enough to avoid that tables would be wrongly tagged as crossable.

> *"The interaction is for sure aided by the system, but requires knowledge of the subject [database domain] and a bit of technical knowledge [relational databases' concepts]"*

Furthermore, despite understanding the relationship graph, some of them were initially confused by why they could choose attributes from more than one table (i.e.,the tables in relationship with the one they were working on) while setting the conversational qualifiers.

Third theme: The current tooltips do not provide satisfactory or clear enough explanation of some tasks
Many users expressed confusions in different points of the tests. The tooltips were often regarded as not clear enough. In some instances users highlighted the need for further technical clarifications, such as whether they had to separate the aliases in the first pattern with commas or not. Or whether in multiple choices the attributes were added in the pattern with an AND or OR logic.

> *"The workflow is good, but what should be written is not explained well enough the first time you interact with the tool"*

> *"Some tooltips should be rephrased to make everything more usable and immediate"*

Another difficulty was encountered while completing task 5 and filling the pattern "Find <table name> that [...]. Users had troubles understanding how to structure the pattern (i.e.,writing a keyword followed by a column they selected).

> *"Can I write 'find foreign professors'?"*

This issue was partially resolved by changing the explanation and adding labels to the text fields, but non-programmers found the task a bit challenging still. For them, an additional layer of complexity was understanding that they could select attributes from tables in a relationship with the one they were working on.

Fourth theme: Perceived user friendliness despite issues with some tasks
During the interviews the majority of the users encountered difficulties with some tasks; despite that, many of them regard the tool as user friendly, noting that the workload is not overwhelming.

> *"If you can resolve all the bugs it would be user friendly"*

> *"It is simple and linear. I do not have too many things to do at once"*

> *"Very usable, [the tasks] flow well"*

Fifth theme: The possibility of generating chatbots from databases is interesting
Users from the first two groups found the possibility of generating a chatbot starting from a relational database interesting. They expressed a positive opinion about the framework, as it enables them to choose flexibly what tables to choose.

> *"It is particularly powerful that the user has control on what tables to include and what relationships to use"*

> *"It is nice to be able to define database queries through natural language"*

7.4.5. Takeaways

This evaluation session was focused on assessing the usability and the effectiveness of the interaction paradigm. The results were mostly positive as the majority of the participants found the interface quite intuitive despite having encountered some difficulties, confirming that the possibility of building a chatbot upon a database is of interest. Furthermore, users from both skill-level groups agreed that to be able to use the tool one must have a clear understanding of the database domain, otherwise it is hard to formulate possible query patterns and complete the annotations. In addition to that, the group of non-programmers, while thinking that the overall interaction is suited for their skill level, expressed dissatisfaction with how the crossable role was explained. In general for this group of users, the concepts

pertaining relational databases proved to be the main obstacle in the interaction with the tool. Most of the participants appreciated the front end, expressing a general positive opinion regarding the interaction paradigm; to most of them, the visual metaphor was clear and easy to understand. The database schema was regarded as the most intuitive window by both programmers and non-programmers. The conversation editor, instead, received mixed feedback, as users from both groups encountered almost the same issues while interacting with it. However the problems highlighted by the participants did not pertain to the interaction paradigm itself, but some specific tasks. Regarding the tasks the main takeaways were:

Multiple renaming operations are hard to understand
Especially non-programmers, that are not familiar with chatbot development, found the reason behind the fact that one can rename the attributes and the tables both in the database schema and in the conversation editor hard to understand. The two renaming operations affect the generation of two different JSON files, the schema and the concept file respectively, but at UI level this distinction is not explicitly declared by choice, as it would add another layer of complexity for a non-programmer. In future work this problem should be addressed either by perfecting the text that explains the two activities, or by unifying the two activities into one so that users will be asked to rename the attributes only once.

Order by and Display annotation needs to be redesigned
The first pattern, "Show me <table name>" was the most critical for the participants. Most of them did not understand the difference between selecting the attributes that have to appear in the answers (display attributes annotation) and selecting those that will be used when results are returned as a list (order by and show columns annotation). In future work this issue could be resolved by separating the operations; leaving the display attributes annotation in the "Mostrami <table name>" pattern, while moving the selection of the attributes that will be shown in a list to another pattern.

Usability refinements
As the web front-end is still in development, usability refinements and additions are needed for a better user experience. Many users expressed dissatisfaction with the explanations of some tasks, in particular with the explanation of the two patterns dedicated to the conversational qualifier annotation in task 5 and 6, which were found too similar. In future work the two patterns need to be modified in order to be more distinct from one another by changing both the pattern and the explanation in pattern 6 (instead of "find <table name> [...]" the pattern should be "find [...]"). Furthermore, users noted that the system should alert them when they are about to do an action that could delete their work. In addition to that, the participants noted that there should be a way to unset the role of a table. Moreover, it needs to be made more clear that the tables listed in the conversational qualifier pattern are tables in relationship with the one the user is working on, because users from all skill levels expressed doubts about this.

Chat preview improvements
While it was generally well received, some users expressed dissatisfaction with the chat preview panel, as its usage was limited. To improve the preview, some users suggested adding the possibility of typing queries to try directly the annotations made. Furthermore, as of now the chat preview shows only the pattern the user is working on. Instead, some participants suggested keeping the whole history of annotations in the preview and, lastly, making its graphical elements more similar to the actual chatbot. In future work all these suggestions could be addressed, in particular the one about keeping the history as it can be used as context.

User friendly but not yet beginner friendly
Many users across all groups expressed a positive judgement regarding the usability of the tool. However, they also agreed that the tool still requires basic knowledge of relational databases. Users from the non-programmers group, in fact, observed that they could use the tool better if they were more knowledgeable about databases and that as of now, while the interaction paradigm is intuitive enough, the tool seems more suitable for more experienced users. Nonetheless, although not statistically comparable with the scores recorded for the other two groups, their score for the SUS questionnaire was still above average
($\bar{x}$ = 74.4, SD = 19.5). Furthermore the NASA-TLX analysis recorded a workload score of ($\bar{x}$ = 35.6, SD = 19.1.).

7.5. About Designing UIs for Conversation Design

The insights gained during the user test brought to light more general key-points about designing EUD environments and related interaction paradigms for conversation design, which can also be interpreted in the light of notable EUD requirements studied and validated in different user-centric studies (Green & Petre, 1996); (Burnett et al., 2004); (Lieberman et al, 2004); (Cappiello et al., 2015).

Closeness of mapping (Lieberman et al, 2004); (Cappiello et al., 2015)
In order to help designers understand the conversation patterns that the system can manage, and the effect that each pattern may have within the conversation flow, it is important to come up with

representations that abstract from technical details and, instead, increase the expressiveness of the roles that each conversation element might play into the final conversation. In other words, the possible types of user queries as well as the related responses that the system can generate needs to be represented in a way that the designers can easily understand. Our choice to "recursively" use a conversational paradigm to design conversations enforces this aspect. The designers express how typical user utterances can be structured, in a format that is exactly the one the final users will adopt for querying the chatbot at execution time. In addition, the preview panel also provides examples of possible responses, thus it immediately reproduces the effects of the designer's choices on the conversation. Designers can thus operate on defining the structure of sentences rather than configuring intents and entities.

Context is king
A UI for conversation design must give users the whole picture of the conversation almost at all times. Knowing how the sentences will appear during conversation, how they relate to one another, and being able to always see previous annotations were, in fact, key concerns for the users. At any step of the conversation design, the designers can take advantage from having an overview on how the conversation would be at run time. In other words, as also demonstrated in other domains where EUD practices are employed (Cappiello et al. 2015), a WYSIWYG (What You See is What You Get) approach, in which the designers can immediately see what the effects of their design decisions are, is beneficial. Previous research on EUD systems (see for example Namoun et al. 2010a, Cappiello et al. 2014), assessed that approaches providing some forms of immediate representation of the design choices increases the user-perceived usefulness and ease of use. For our paradigm, a preliminary confirmation of this benefit is highlighted by the scores of the NASA-TLX and SUS questionnaires for the three categories of users involved in our final study.

Progressive evaluation (Lieberman et al. 2006); (Cappiello et al. 2015)
To enhance the designers' perception of and the control on the effects that the pattern configurations have on the final conversation, it is important to provide feedback on "how the designer is doing". Since the very first configuration action, the designer must be enabled to see a running application and to observe incrementally the effect of any other subsequent configuration action. Immediate execution paradigms, based on WYSIWYG representations, where each single configuration action is observable through the immediate execution of the configured patterns, are therefore good candidates to support progressive evaluation. These mechanisms also avoid the so-called premature commitment, because the user is not forced to make decisions without being able to observe and evaluate the effect of such decisions.

Assistance on intent and entity identification.
To further smooth design barriers, and to let the designers define meaningful conversations, it is important to provide assistance in the identification of intents and entities. This is required especially to assist those designers who do not have sufficient development knowledge in general (e.g., non-programmers), or in the specific field of chatbots (e.g., programmers not used to develop chatbots). In the approach presented in this thesis, conversation design is assisted by providing pre-defined patterns of queries that correspond to typical intents for data exploration. The entities on which these patterns focus can then be easily identified starting from the database tables. The designers can still control the configuration of intents and entities: they can specify entity names and their aliases, and can define typical expressions that the final users could use during a conversation to refer to specific intents. An additional aspect that can improve system assistance, not covered in this thesis but already planned among the future extensions, relates to the provision of default configurations (i.e., suggestions on possible settings for some conversational patterns), which can be adopted as they are, or adequately modified. These suggestions, for example the one on categorical attributes, can be identified by analysing the properties of the database schema and of the database content; this is indeed another advantage that the integration of the dialogue system with the database can offer.

8. Conclusions

8.1. Summing Up

In this thesis, an end-user development tool was proposed for the CHATIDEA framework to make the annotation procedure more accessible to non-programmers and, more in general, to users that are not familiar with the framework. The research started with analysing the state-of-the-art of chatbot development platforms. What was found is that, as chatbots for data exploration are still a new concept, the platforms available right now lack in general a way to directly query databases. Moreover, even though some of them can be integrated with a database, this operation is still handled by code integrations that, thus, requires programmers to be executed. On this premise, a graphical user interface was designed to provide a novel tool to enrich CHATIDEA, the already existing framework for the development of conversational agents for data exploration. Then, before presenting the proposed interface, the requirements of such a tool were outlined, addressing also some of the challenges posed by the annotation procedure characteristics. After that, the conversational paradigm used to present the annotation concepts to the user was discussed, highlighting how the familiar environment provided by such a paradigm could be leveraged to help users in understanding and contextualising the data and tasks they are called to complete. Then, the project itself was discussed, explaining its information architecture, layout and visual identity. Finally, a user-based evaluation was conducted to validate the design choices. These evaluation sessions allowed us to find issues in the design that led to significant improvements, but also highlighted some critical issues that will be addressed in future work. Nonetheless, the evaluation assessed a general positive feedback regarding the interface, that, despite the limitations, was regarded as fairly intuitive and accessible even for a beginner audience provided that the user has a basic understanding of the content of the database and of the structure of relational databases.

8.2. Limitations

At the moment, the interface expects the user to make all the annotations, further iteration of the system could allow users to not make some annotations that are not fundamental to the functioning of the chatbot, such as the category annotation. Furthermore, the system does not cover one of the additions presented in (Ferreri & Notari, 2020), the similar JSON file. The file contains a group classification of similar attributes present in different tables, for example, one group could contain every attribute that refers to people's names. So, if we have two tables, one for journalists and one for winners of the Pulitzer prize, a query containing one of these two attributes would be ambiguous, because the chatbot will not know which table to look into first to get the name the user searched for. The similar annotation makes it so that the chatbot will ask the user for clarification before returning an answer.
In addition to that, as emerged during the second user evaluation, the system could prove to be difficult to use with very large databases, as it would be increasingly difficult to keep track of all the relationships and annotations. In the third evaluation study, the participants from the non-programmers group highlighted that as of now, the system still requires some knowledge of relational databases, and, thus, is not perfectly suited for their skill level. Furthermore, all the participants in the study had troubles with specific tasks that will need to be redesigned in future work.

8.3. Further Developments

The interface could benefit from automation of certain procedures to simplify the workflow. The interface, in fact, still requires a certain knowledge of relational databases, that makes it still not completely suitable for totally inexperienced users.

Improvements regard in particular the drawing of new relationships and the tagging of the type of tables. To help users in drawing new relationships, after the user has selected a primary key, the system could highlight automatically the other tables containing the corresponding foreign key, so that the user has a visual cue to where to draw the node. Regarding the type tagging, instead, an algorithm could be developed to automatically tag the crossable tables as crossable. This would remove the knowledge obstacle posed by the crossable tables and make the tool more accessible to users that lack a basic knowledge of relational databases.

Another step that could be automated is the category annotation, by defining an algorithm that could automatically

understand from the data which attribute is the best candidate to represent the data in a chart.

Lastly the interaction that generates the Show columns and order by annotation needs to be redesigned. Possibly by separating it from the pattern “show me <table name>” and by defining a new pattern.

8.4. Publication

The interaction paradigm and GUI proposed in this thesis, together with the methodology, i.e., the conceptual modelling of conversational elements by annotation of the database schema, will be published in the following article:

> Ludovica Piro, Giuseppe Desolda, Maristella Matera, Sara Mosca, 2020, *A Converational Paradigm for the EUD of Chatbots for Data Exploration*.

The article will be submitted to the 18th International Conference on Human–Computer Interaction — Interact — to be held on July 2021.

9. Bibliography

Akcora, E., Belli, A., Berardi, M., Casola, S., Di Blas, N., Falletta, S., Faraotti, A., Lodi, L., Diaz, D., Paolini, P., Renzi, F., & Vannella, F. (2018). Conversational Support for Education (pp. 14–19). https://doi.org/10.1007/978-3-319-93846-2_3

Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). Natural Language Interfaces to Databases—An Introduction. *ArXiv:Cmp-Lg/9503016*. http://arxiv.org/abs/cmp-lg/9503016

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. Qualitative Research in Psychology, 3, 77–101. https://doi.org/10.1191/1478088706qp063oa

Brooke, J. (1995). SUS: A quick and dirty usability scale. Usability Eval. Ind., 189.

Brooks, A. (2018, November 1). 10 Best Chatbot Builders in 2021. *Venture Harbour*. https://www.ventureharbour.com/best-chatbot-builders/

Burnett, M., Cook, C., & Rothermel, G. (2004). End-user Software Engineering. Commun. ACM, 47, 53–58. https://doi.org/10.1145/1015864.1015889

Cahn, J. (2017). *CHATBOT: Architecture, Design, & Development*. University of Pennsylvania.

Canh, N. T. (2018, May 4). *Turn your database into a chatbot*. Medium. https://medium.com/botfuel/turn-your-database-into-a-chatbot-10dae003b97d

Cassell, J., Bickmore, T., Campbell, L., & Vilhjlmsson, H. (2000). Conversation as a System Framework: Designing Embodied Conversational Agents. *Embodied Conversational Agents*.

Cappiello, C., Matera, M., & Picozzi, M. (2015). A UI-Centric Approach for the End-User Development of Multidevice Mashups. ACM Transactions on the Web, 9(3), 11:1-11:40. https://doi.org/10.1145/2735632

Castaldo, N. (2019). *A conceptual modeling approach for the rapid development of chatbots for conversational data exploration* [Politecnico di Milano]. http://hdl.handle.net/10589/147420

Chen, H., Liu, X., Yin, D., & Tang, J. (2017). A Survey on Dialogue Systems: Recent Advances and New Frontiers. *ACM SIGKDD Explorations Newsletter*, *19*(2), 25–35. https://doi.org/10.1145/3166054.3166058

China, WeChat, and the Origins of Chatbots | by Jerry @ Rocketbots.io | Chatbots Magazine. (n.d.). Retrieved 13 March 2021, from https://chatbotsmagazine.com/china-wechat-and-the-origins-of-chatbots-89c481f15a44

Conway, M., Audia, S., Burnette, T., Cosgrove, D., Christiansen, K., Deline, R., Durbin, J., Gossweiler, R., Koga, S., Long, C., Mallory, B., Miale, S., Monkaitis, K., Patten, J., Pierce, J., Shochet, J., Staack, D., Stearns, B., Stoakley, R., & Pausch, R. (2000). *Alice: Lessons Learned from Building a 3D System For Novices*.

Everest, G. (1976). Basic Data Structure Models Explained With A Common Example. Computing Systems, 39–46.

Danescu-Niculescu-Mizil, C., & Lee, L. (2011). Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. *ArXiv:1106.3077 [Physics]*. http://arxiv.org/abs/1106.3077

Ferreri, E., & Notari, D. R. (2020). *Designing and validating conversational agents for data exploration*. [Politecnico di Milano]. http://hdl.handle.net/10589/165068

Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., Schlaefer, N., & Welty, C. (2010). Building Watson: An Overview of the DeepQA Project. *AI Magazine*, *31*(3), 59–79. https://doi.org/10.1609/aimag.v31i3.2303

Green, T. R. G., & Petre, M. (1996). Usability Analysis of Visual

Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages & Computing*, *7*(2), 131–174. https://doi.org/10.1006/jvlc.1996.0009

Hart, S. G., & Staveland, L. E. (1988). Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In P. A. Hancock & N. Meshkati (Eds.), Advances in Psychology (Vol. 52, pp. 139–183). North-Holland. https://doi.org/10.1016/S0166-4115(08)62386-9

Hussain, S., Sianaki, O., & Ababneh, N. (2019). *A Survey on Conversational Agents/Chatbots Classification and Design Techniques* (pp. 946–956). https://doi.org/10.1007/978-3-030-15035-8_93

Janarthanam, S. (2017). *Hands-On Chatbots and Conversational UI Development: Build chatbots and voice user interfaces with Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio, and Alexa Skills*. Packt Publishing Ltd.

Khan, R., & Das, A. (2017). *Build Better Chatbots: A Complete Guide to Getting Started with Chatbots*. Apress.

Klopfenstein, L., Delpriori, S., Malatini, S., & Bogliolo, A. (2017). *The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms*. 555–565. https://doi.org/10.1145/3064663.3064672

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., & Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys*, *43*(3), 21:1-21:44. https://doi.org/10.1145/1922649.1922658

Leong, M.-K. (2004). Conversational Design as a Paradigm for User Interaction on Mobile Devices. In F. Crestani, M. Dunlop, & S. Mizzaro (Eds.), *Mobile and Ubiquitous Information Access* (pp. 11–27). Springer. https://doi.org/10.1007/978-3-540-24641-1_2

Lewis, J. R., & Sauro, J. (2009). The Factor Structure of the System Usability Scale. In M. Kurosu (Ed.), Human Centered Design (pp. 94–103). Springer. https://doi.org/10.1007/978-3-642-02806-9_12

Li, F., & Jagadish, H. V. (2014). NaLIR: An interactive natural language interface for querying relational databases. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 709–712. https://doi.org/10.1145/2588555.2594519

Lieberman, H., Paternò, F., & Wulf, V. (Eds.). (2006). *End-User Development* (1.ed., 2. printing). Springer.

Marcus, A. (1998). Metaphor design in user interfaces. *ACM Sigdoc Asterisk Journal of Computer Documentation*, *22*, 129–130. https://doi.org/10.1145/286498.286577

Mauldin, M. L. (1994). ChatterBots, TinyMuds, and the Turing test: Entering the Loebner Prize competition. *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, 16–21.

McConnell, S. (2004). *Code Complete*. Pearson Education.

McKitterick, W. (2016, April 23). *Messaging apps are now bigger than social networks*. Business Insider. https://www.businessinsider.com/the-messaging-app-report-2016-4-23

Mindbowser. (2017). Chatbot Survey 2020. *Mindbowser*. https://www.mindbowser.com/industry-report/chatboot-survey/

Monsters Data. (2019, November 29). *50 Chatbot Platforms: 2019 Edition*. Medium. https://medium.com/@datamonsters/50-chatbot-platforms-2019-edition-e6be022a7e0e

Myers, B. (1990). Creating User Interfaces Using Programming by Example, Visual Programming, and Constraints. *ACM Trans. Program. Lang. Syst.*, *12*, 143–177. https://doi.org/10.1145/78942.78943

Namoun, A., Nestler, T., & De Angeli, A. (2010). Service Composition for Non-programmers: Prospects, Problems, and Design Recommendations. 123–130. https://doi.org/10.1109/ECOWS.2010.17

Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing* (1st ed.). MIT Press.

Nickerson, R. S. (1976). On conversational interaction with computers. *Proceedings of the ACM/SIGGRAPH Workshop on User-Oriented Design of Interactive Graphics Systems*, 101–113. https://doi.org/10.1145/1024273.1024286

Nuruzzaman, M., & Hussain, O. K. (2020). IntelliBot: A Dialogue-based chatbot for the insurance industry. *Knowledge-Based Systems*, *196*, 105810. https://doi.org/10.1016/j.knosys.2020.105810

Patil, P. (2020, July 17). *Top 30 Powerful and Best Platforms To Build Chatbots*. Medium. https://chatbotsjournal.com/top-30-powerful-and-best-platforms-to-build-chatbots-

bf413419d584

Peng, Z., & Ma, X. (2019). A survey on construction and enhancement methods in service chatbots design. *CCF Transactions on Pervasive Computing and Interaction*, *1*(3), 204–223. https://doi.org/10.1007/s42486-019-00012-3

Pereira, J., & Díaz, O. (2018). Chatbot Dimensions that Matter: Lessons from the Trenches (pp. 129–135). https://doi.org/10.1007/978-3-319-91662-0_9

Rehan, A. (2018, October 3). *10 Best Chatbot Development Frameworks to Build Powerful Bots*. Geekflare. https://geekflare.com/chatbot-development-frameworks/

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Ritter, A., Cherry, C., & Dolan, B. (2010). Unsupervised Modeling of Twitter Conversations. *Proceedings of HLT-NAACL*.

Roller, S., Dinan, E., Goyal, N., Ju, D., Williamson, M., Liu, Y., Xu, J., Ott, M., Shuster, K., Smith, E. M., Boureau, Y.-L., & Weston, J. (2020). Recipes for building an open-domain chatbot. *ArXiv:2004.13637 [Cs]*. http://arxiv.org/abs/2004.13637

Sauro, J. (2011, February 11). MeasuringU: Measuring Usability with the System Usability Scale (SUS). https://measuringu.com/sus/

Turing, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, *LIX*(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433

Vanhemert, K. (2015, June 26). The Future of UI Design? Old-School Text Messages. *Wired*. https://www.wired.com/2015/06/future-ui-design-old-school-text-messages/

Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45. https://doi.org/10.1145/365153.365168

Whitehorn, M., & Marklyn, B. (Eds.). (2007). Tables. In *Inside Relational Databases with Examples in Access* (pp. 17–35). Springer. https://doi.org/10.1007/978-1-84628-687-2_3

Woods, W., Kaplan, R., & Webber, B. (1972). *The Lunar Science Natural Language Information System: Final Report*.

Wouters, J. (n.d.). Chatbot Platform Comparison | Compare 39 chatbot builders. *Chatimize*. Retrieved 22 March 2021, from https://chatimize.com/chatbot-platform-comparison/

Yaghoub-Zadeh-Fard, M., Benatallah, B., Casati, F., Barukh, M. C., & Zamanirad, S. (2020). User Utterance Acquisition for Training Task-Oriented Bots: A Review of Challenges, Techniques and Opportunities. IEEE Internet Computing, 24(3), 30–38. https://doi.org/10.1109/MIC.2020.2978157