Dialogue Patterns and Composite Intents Recognition in Task-oriented Human-Chatbot Conversations

Sara Bouguelia

THÈSE de DOCTORAT DE
L'UNIVERSITÉ CLAUDE BERNARD LYON 1

École Doctorale N° 512
Mathématiques et Informatique (InfoMaths)

Discipline: Informatique

Soutenue publiquement le 27/06/2023 par :
Sara Bouguelia

Modèles de Dialogue et Reconnaissance d'Intentions Composites dans les Conversations Utilisateur-Chatbot orientées Tâches

---

Devant le jury composé de :

Mme. Daniela GRIGORI	Professeur	Université Paris-Dauphine	Rapporteure
Mme. Lynda TAMINE	Professeur	Université Paul Sabatier	Rapporteure
M. Khalid BENABDESLEM	MCF - HDR	Université Lyon 1	Examinateur
Mme. Salima BENBERNOU	Professeur	Université Paris Descartes	Présidente
Mme. Barbara PERNICI	Professeur	École polytechnique de Milan	Examinatrice
M. Boualem BENATALLAH	Professeur	Université de Dublin (DCU)	Co-directeur
M. Hamamache KHEDDOUCI	Professeur	Université Lyon 1	Co-directeur

Université Claude Bernard – LYON 1

Président de l'Université	M. Frédéric FLEURY
Président du Conseil Académique	M. Hamda BEN HADID
Vice-Président du Conseil d'Administration	M. Didier REVEL
Vice-Président du Conseil des Etudes et de la Vie Universitaire	M. Philippe CHEVALLIER
Vice-Président de la Commission de Recherche	M. Petru MIRONESCU
Directeur Général des Services	M. Pierre ROLLAND

COMPOSANTES SANTE

Département de Formation et Centre de Recherche en Biologie Humaine	Directrice : Mme Anne-Marie SCHOTT
Faculté d'Odontologie	Doyenne : Mme Dominique SEUX
Faculté de Médecine et Maïeutique Lyon Sud - Charles Mérieux	Doyenne : Mme Carole BURILLON
Faculté de Médecine Lyon-Est	Doyen : M. Gilles RODE
Institut des Sciences et Techniques de la Réadaptation (ISTR)	Directeur : M. Xavier PERROT
Institut des Sciences Pharmaceutiques et Biologiques (ISBP)	Directrice : Mme Christine VINCIGUERRA

COMPOSANTES & DEPARTEMENTS DE SCIENCES & TECHNOLOGIE

Département Génie Electrique et des Procédés (GEP)	Directrice : Mme Rosaria FERRIGNO
Département Informatique	Directeur : M. Behzad SHARIAT
Département Mécanique	Directeur M. Marc BUFFAT
Ecole Supérieure de Chimie, Physique, Electronique (CPE Lyon)	Directeur : Gérard PIGNAULT
Institut de Science Financière et d'Assurances (ISFA)	Directeur : M. Nicolas LEBOISNE
Institut National du Professorat et de l'Education	Administrateur Provisoire : M. Pierre CHAREYRON
Institut Universitaire de Technologie de Lyon 1	Directeur : M. Christophe VITON
Observatoire de Lyon	Directrice : Mme Isabelle DANIEL
Polytechnique Lyon	Directeur : Emmanuel PERRIN
UFR Biosciences	Administratrice provisoire : Mme Kathrin GIESELER
UFR des Sciences et Techniques des Activités Physiques et Sportives (STAPS)	Directeur : M. Yannick VANPOULLE
UFR Faculté des Sciences	Directeur : M. Bruno ANDRIOLETTI

Abstract

Dialogue Systems (or simply *chatbots*) are in very high demand these days. They enable the understanding of user needs (or *user intents*), expressed in natural language, and on fulfilling such intents by invoking the appropriate back-end APIs (Application Programming Interfaces). Chatbots are famed for their easy-to-use interface and gentle learning curve (it only requires one of humans' most innate ability, the use of *natural language*). The continuous improvement in Artificial Intelligence (AI), Natural Language Processing (NLP), and the countless number of devices allow performing real-world tasks (e.g., making a reservation) by using natural language-based interactions between users and a large number software enabled services.

Nonetheless, chatbot development is still in its preliminary stage, and there are several theoretical and technical challenges that need to be addressed. One of the challenges stems from the wide range of utterance variations in open-end human-chatbot interactions. Additionally, there is a vast space of software services that may be unknown at development time. Natural human conversations can be rich, potentially ambiguous, and express complex and context-dependent intents. Traditional business process and service composition modeling and orchestration techniques are limited to support such conversations because they usually assume a priori expectation of what information and applications will be accessed and how users will explore these sources and services. Limiting conversations to a process model means that we can only support a small fraction of possible conversations. While existing advances in NLP and Machine Learning (ML) techniques automate various tasks such as intent recognition, the synthesis of API calls to support a broad range of potentially complex user intents is still largely a

manual, ad-hoc and costly process.

This thesis project aims at advancing the fundamental understanding of cognitive services engineering. In this thesis we contribute novel abstractions and techniques focusing on the synthesis of API calls to support a broad range of potentially complex user intents. We propose reusable and extensible techniques to recognize and realize complex intents during humans-chatbots-services interactions. These abstractions and techniques seek to unlock the seamless and scalable integration of natural language-based conversations with software-enabled services.

Résumé

Les Systèmes de Dialogue (ou simplement *chatbots*) sont très demandés de nos jours. Ils permettent de comprendre les besoins des utilisateurs (ou *intentions des utilisateurs*), exprimés en langage naturel, et de répondre à ces intentions en invoquant les APIs (Interfaces de Programmation d'Application) appropriées. Les chatbots sont connus pour leur interface facile à utiliser et ils ne nécessitent que l'une des capacités les plus innées des humains qui est l'utilisation du *langage naturel*. L'amélioration continue de l'Intelligence Artificielle (IA), du Traitement du Langage Naturel (NLP) et du nombre incalculable de dispositifs permettent d'effectuer des tâches réelles (par exemple, faire une réservation) en utilisant des interactions basées sur le langage naturel entre les utilisateurs et un grand nombre de services.

Néanmoins, le développement de chatbots est encore à un stade préliminaire, avec plusieurs défis théoriques et techniques non résolus découlant de (i) la variations d'énoncés dans les interactions humain-chatbot en libre échange et (ii) du grand nombre de services logiciels potentiellement inconnus au moment du développement. Les conversations en langage naturel des personnes peuvent être riches, potentiellement ambiguës et exprimer des intentions complexes et dépendantes du contexte. Les techniques traditionnelles de modélisation et d'orchestration de processus et de composition de services sont limitées pour soutenir de telles conversations car elles supposent généralement une attente a priori de quelles informations et applications seront accédées et comment les utilisateurs exploreront ces sources et services. Limiter les conversations à un modèle de processus signifie que nous ne pouvons soutenir qu'une petite fraction de conversations possibles. Bien que les avancées existantes dans les techniques de

NLP et d'apprentissage automatique (ML) automatisent diverses tâches telles que la reconnaissance d'intention, la synthèse d'appels API pour prendre en charge une large gamme d'intentions d'utilisateurs potentiellement complexes est encore largement un processus manuel et coûteux.

Ce projet de thèse vise à faire avancer la compréhension fondamentale de l'ingénierie des services cognitifs. Dans cette thèse, nous contribuons à des abstractions et des techniques novatrices axées sur la synthèse d'appels API pour soutenir une large gamme d'intentions d'utilisateurs potentiellement complexes. Nous proposons des techniques réutilisables et extensibles pour reconnaître et réaliser des intentions complexes lors des interactions entre humains, chatbots et services. Ces abstractions et techniques visent à débloquer l'intégration transparente et évolutive de conversations basées sur le langage naturel avec des services activés par logiciel.

Contents

List of Figures

List of Tables

Publications

Conferences

- **Sara Bouguelia**, Auday Berro, Boualem Benatallah, Marcos Báez, Hayet Brabra, Shayan Zamanirad, Hamamache Kheddouci, "Process-Oriented Intents: A Cornerstone for Superimposition of Natural Language Conversations over Composite Services," in ICSOC, 2022.
- **Sara Bouguelia**, Hayet Brabra, Boualem Benatallah, Marcos Baez, Shayan Zamanirad, Hamamache Kheddouci, "Context knowledge-aware recognition of composite intents in task-oriented human-bot conversations," in CAiSE, 2022 (**Best Paper Award**).
- **Sara Bouguelia**, Hayet Brabra, Shayan Zamanirad, Boualem Benatallah, Marcos Baez, Hamamache Kheddouci, "Reusable abstractions and patterns for recognising compositional conversational flows," in CAiSE, 2021.
- Shayan Zamanirad, Boualem Benatallah, Carlos Rodriguez, Mohammadali Yaghoubzadehfard, **Sara Bouguelia**, Hayet Brabra, "State machine based human-bot conversation model and services," in CAiSE, 2020.

Journals

- Hayet Brabra, Marcos Báez, Boualem Benatallah, Walid Gaaloul, **Sara Bouguelia**, Shayan Zamanirad, "Dialogue management in conversational systems: a review of approaches, challenges, and opportunities," in IEEE Transactions on Cognitive and Developmental Systems (TCDS), 2021.

Chapter 1

Introduction

Contents

1.1 Background, Motivations and Aims

In the past, the notion of having a virtual assistant or chat companion system with adequate intelligence seemed like a far-fetched concept that only existed in the realm of science fiction. However, with the emergence of conversational Artificial Intelligence

(AI) and its instantiation in the form of messaging, this concept is no longer an illusion [49,112]. The development of natural language processing (NLP) and machine learning (ML) techniques has paved the way for the creation of human-computer conversation systems that can act as our personal assistant or chat companion [49,53,258]. Building *dialogue systems*, also known as *virtual assistants*, *conversational agents*, *conversational AI* or simply *chatbots*, has become increasingly relevant in facilitating human-computer interactions. These conversational AI based services enable the understanding of user needs, expressed in natural language (text or voice), and on fulfilling such needs by invoking the appropriate back-end services [248].

Dialogue systems have attracted increasing attention due to their promising potentials and commercial values. Conversational AI is already recognized as a strategic priority for modern enterprises [115]. Increasingly organizations have started or plan to use capabilities arising from advances in conversational AI. This has led to the rise of virtual personal assistants and sophisticated chatbots, which have become an integral part of our daily lives. With popular personal assistants like Microsoft Cortana, Google Assistant, Amazon Alexa, and Apple Siri being used by millions of users worldwide [62], chatbots have gained immense popularity. Apart from the popular personal assistants, many other chatbots are being developed, such as those allowing data scientists to assemble data analytic pipelines (e.g., Analyza [66]) and those that act like humans (e.g., Replika [93]). These chatbots have found applications in different domains such as healthcare, finance, marketing, and customer support. We can use them for generating source code, controlling home appliances, and even testing theories [37,71,75,141]. At the time of writing, the company *OpenAI* [12] has developed a new chatbot, called *chatGPT* [3], which has opened up new possibilities and revolutionize the way we communicate and learn, making it easier and faster to access information and get the help we need [117,144]. In less than 2 months, the artificial intelligence ChatGPT has reached 100 million monthly active users, making it the fastest-growing consumer application in history [68,117,144]. Its success is a testament to the importance and potential of chatbots in today's and future world.

Nonetheless, despite the advances in various research areas and the greater availabil-

ity of data and services, developing task-oriented chatbots remains a challenging task. One of the key challenges is the translation of user utterances to intents. Human language can be rich, potentially ambiguous, and express ambiguous and complex intents, which poses a difficulty for chatbots to understand user requests accurately [37,71,108]. Furthermore, with the continuous development of software-enabled services and the increasing availability of new APIs (Application Programming Interfaces), chatbot development needs to scale in terms of how effectively they can be integrated with APIs. However, from an engineering perspective, the integration of chatbots and software-enabled services has not kept pace with the ability to deploy individual devices and services [244]. The lack of effective support for a wide range of possibly complex user intents and the inability to leverage the large and growing number of services hinder the design and development of effective techniques that allow users to interact naturally with software-enabled services. Current solutions for chatbot development rely heavily on experts' understanding of APIs and on the availability of massive amounts of annotated data. Using existing NLP and ML techniques for recognizing complex user intents will require laborious, costly and hard-to-acquire training datasets. In addition, each time a new complex intent is identified, extending or producing a new dataset is needed as well. Therefore, more advanced and flexible techniques are required to cater for complex intent recognition and chatbot development. After all, the value of conversational AI heavily depends on their ability to easily integrate and reuse concomitant capabilities across a large number of heterogeneous and evolving APIs [244, 248].

In this thesis, we contribute novel abstractions and techniques focusing on the synthesis of API calls to support broad range of potentially complex user intents. We propose reusable and extensible techniques to recognize and realize composite intents during humans-chatbots-services interactions. The abstractions and techniques seek to unlock the seamless and scalable integration of natural language-based conversations with software-enabled services.

1.2 Research Issues

In this section, we will discuss important issues regarding the management of complex and composite interactions between users, chatbots, and services.

1.2.1 Access to heterogeneous information sources

Dialogue systems require access to information stored in various data sources and services in order to fulfill user intent and provide relevant responses. This includes information from conversation history, user profiles, and external sources such as documents [108]. Such information is in many cases scattered across different, heterogeneous and complex information silos [53, 227]. While existing techniques in information retrieval and indexing [76, 230, 234] have produced promising results that are certainly useful, more advanced techniques that cater for managing information and complex interactions between users, chatbots, and services are needed to effectively support users requests.

A core challenge is the lack of latent knowledge about user intents and APIs (e.g., relationships between intents and APIs elements), which makes it hard to effectively support dynamic synthesis of services and reason about potentially complex user intents (e.g., ambiguous natural language user utterances, context-specific user tasks). Incorrect inference of conversation flows arises from uncertainty about intent slot values and relationship between API elements across heterogeneous APIs (e.g., one intent uses *city* as a parameter while another use *location* as a parameter). In this context, there is a need for more advanced techniques that focus on augmenting intents with knowledge that facilitates the superimposition of natural language conversations over process and software-enabled services to effectively support users' requests.

1.2.2 Support complex users-chatbots-services interactions

Developing chatbots capable of handling complex conversations spanning multiple topics and domains is an ongoing area of research. There are still major challenges, including that building such chatbots requires rich abstractions to capture user intents that may be context-depends and complex [37, 71]. For instance, the realization of a user intent may require composition of multiple APIs to control IoT (Internet of Things) devices using one user utterance. Imagine a scenario where a user wants to go to sleep and says *"I am going to sleep"* to the chatbot. The chatbot recognizes this as a composite intent and triggers multiple APIs to perform the necessary actions, such as turning off the TV, lights, and any other devices that may interfere with sleep. The chatbot could also adjust the temperature and humidity levels, set an alarm for the next morning, and lock the doors for security. This example illustrates the potential of chatbots to provide context-aware assistance in managing complex tasks, but it also highlights the challenges of integrating multiple APIs and recognizing composite intents.

While existing advances in NLP and ML techniques have made significant progress in automating various tasks such as intent recognition [50], the synthesis of API calls to support broad range of potentially composite user intents remains largely a manual and costly process [247]. In addition, using these techniques require massive amounts and hard to acquire training datasets. The latent knowledge required to integrate intents and APIs until now is rarely codified and used to recognize composite intents and translate them into APIs and their compositions. Therefore, there is a need for more advanced and flexible techniques that cater for composite intent recognition to build robust virtual assistants and provide a better user experience.

1.3 Contributions

Our main goal is to support composite user intents by dynamically and incrementally synthesizing executable conversation models from natural language conversations. To achieve this, we build upon advances in ML and NLP techniques, as well as conver-

sation modeling and knowledge graph approaches. We contribute innovative concepts and techniques to enable automatic recognition and realization of composite and complex user intents during natural language conversations with software enabled services. The proposed concepts and techniques include: (i) reusable abstractions and patterns for recognizing compositional conversational flows, (ii) context knowledge-aware recognition of composite intents in task-oriented human-bot conversations, and (iii) process-oriented intents for superimposition of natural language conversations over composite services. We investigate and develop software architectures, prototypes, and evaluation studies to assess the proposed models and techniques.

1.3.1 Reusable Abstractions and Patterns for Recognizing compositional conversational flows

Natural user conversations can be rich, potentially ambiguous, and express *composite user intents*. The latter may be context-dependent and complex. Therefore, its realization may require the composition of multiple APIs (e.g., triggering multiple APIs to control IoT devices using one user utterance) and sophisticated context management. A key distinguishing feature of task-oriented conversational services is *dialogue patterns*, the multi-turn interaction styles needed to fulfill user intents (e.g., a chatbot-to-user question to resolve the value of a missing intent parameter, an invocation of an API by the chatbot to resolve the value of a missing parameter, or extracting an intent parameter value from the user-chatbot conversation history). Instead of relying on low-level scripting mechanisms to manage interactions, we argue that models for describing natural language interactions between users, chatbot, and services should be endowed with intuitive constructs that can be used to specify a range of dialogue patterns. In [248], we proposed the concept of conversation state machines as an abstraction to represent and reason about dialogue patterns.

In this contribution, we propose to extend the conversation state machines model by identifying and characterizing a set of *composite dialogue patterns* (e.g., data flow between API methods, nested API methods, dependency constraints between API meth-

ods) to identify complex user intents. These composite dialogue patterns endow chatbot platforms with reusable functionality to recognize compositional conversational flows that would otherwise have to be implemented by bot developers (i.e., reduce the development complexity).

1.3.2 Context Knowledge-aware Recognition of Composite Intents in Task-oriented Human-Bot Conversations

Ideally, a virtual assistant should detect user intents and infer missing slot values with the least possible interactions with the user (i.e., the virtual assistant asks the user for a missing value only when it cannot infer it from other sources). A key challenge to achieve this objective is devising robust intent recognition and slot inference despite the potentially ambiguous and complex utterances. An utterance may not always follow a simple conversation pattern, where the chatbot recognizes a *basic intent* and infers all required slot values from the utterance. The user intent can be *composite* and its realization may require the chatbot to break it down into a list of atomic actions and infer potentially missing values from different sources, not directly from the utterance. Existing NLP and ML techniques have produced promising and useful results to recognize basic intents. However, ML based techniques rely on the availability of massive amounts of annotated data. Using these techniques to recognize composite intents requires laborious, costly and hard to acquire training datasets. In addition, each time a new composite intent is identified, extending or producing a new dataset is needed as well. Therefore, more advanced and flexible techniques that cater for composite intent recognition are needed.

In this contribution, we propose reusable and extensible rule-based technique that uses a sophisticated context service to recognize and realize composite intents. We believe that our approach charts novel abstractions that unlock the seamless and scalable integration of natural language-based conversations with software-enabled services. We devise a novel composite intent recognition that allows the incremental acquisition of rule templates to identify composite intents from basic dialogue acts and context

features. The contextual knowledge required at run-time to recognize composite intents and infer slot values from user-chatbot conversations is extracted from conversation history, enriched entities, intents and API schemas and represented in graph structure.

1.3.3 Process-oriented Intents for Superimposition of Natural Language Conversations over Composite Services

Integrating task-oriented conversational assistants and process-enabled automation involves advanced machine learning, entity recognition, and NLP techniques. A key NLP task in this context is intent recognition, i.e., (i) understanding user utterances in natural language and recognizing user intent corresponding to tasks that the user wants to accomplish, (ii) extracting relevant intent input slot values from user utterances, and (iii) trigger commands that process user intents and perform conversations with users. Orchestrating human–machine conversations over *composite services* requires rich abstractions and knowledge to: (i) interact with a multi-step processes using natural language utterances, (ii) automatically recognise nuanced, context sensitive and possibly ambiguous process-aware user intents including starting a new task, inquiring about task progress, switching from one task to another and exceptional behavior such as canceling or updating tasks.

In this contribution, we focus on the superimposition of task-oriented assistants over composite services. Specifically, we identify fine-grained *Human-bot-Process (HP) interaction acts* (or *process-oriented user intents*) that are relevant to represent natural language conversations between the user and multi-step processes. We devise an approach that combines recognition of these process-oriented intents from user utterances with additional context and process knowledge to enable human users to perform tasks by naturally interacting with service orchestrations.

1.4 Thesis structure

The structure of the thesis follows the research approach presented in Figure 1.1 and consists of five chapters. Chapter 2 provides essential background knowledge and reviews state of the art approaches in the field. The next three chapters (Chapter 3, 4, and 5) present the related work to the study reported in each chapter to motivate the research and illustrate its relevance. Specifically, Chapter 3 proposes and presents reusable dialogue patterns for recognizing compositional conversational flows to support increased complexity and expressivity during human-chatbot-services conversations. Chapter 4 proposes and presents a novel approach to recognize and realize composite user intents. Chapter 5 proposes and presents the concept of *Human-bot-Process interaction acts*. These interaction acts are relevant to represent natural language conversations between the user and multi-step processes. Finally, Chapter 6 provides concluding remarks and discusses future research directions.

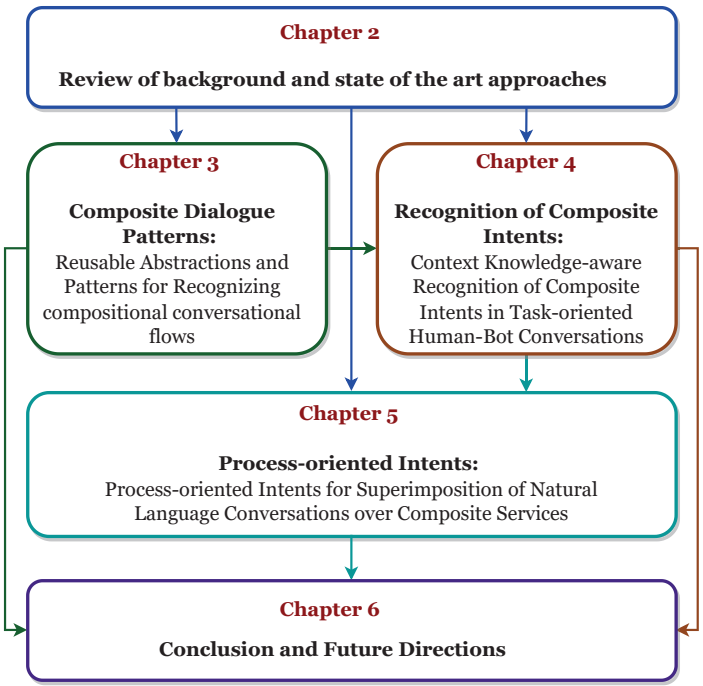

Figure 1.1: Research Approach.

Chapter 2

Background and State of the art

Contents

Dialogue systems have garnered increasing attention in recent years due to their ability to understand and respond to natural language inputs. This chapter is divided into

four sections. In Section 2.1, we discuss the main concepts related to dialogue systems. In Section 2.2, we delve into the state of the art and discuss *dialogue management* techniques. In Section 2.3, we explore different knowledge sources used in task-oriented chatbots for *dialogue state tracking*. In Section 2.4, we summarize dialogue management approaches and knowledge sources in task-oriented chatbots and then we identify some research issues.

2.1 Background

In this section, we discuss the background of dialogue systems. This section is divided into four main parts: In Section 2.1.1, we define dialogue systems and their different types. We then focus on *task-oriented dialogue systems*, which are the focus of this thesis. In Section 2.1.2, we define *intents* and *slots* and present the *natural language understanding* component, which is one of the main components of task-oriented dialogue systems. In Section 2.1.3, we define the *dialogue management* component, which is another important component of task-oriented dialogue systems. Finally, in Section 2.1.4, we provide an overview of techniques used to generate human-like responses, which is the third and final component of task-oriented dialogue systems. Overall, Sections 2.1.2, 2.1.3, and 2.1.4 provide a comprehensive overview of the three components that make up a task-oriented dialogue system.

2.1.1 Dialogue systems

Dialogue systems, also known as *chatbots*, are programs that allow the user to converse with a computer in natural language [34, 49]. These systems take a natural language utterance (in form of text or voice) as input and generate an appropriate natural language response (in form of text or voice) as output [49, 112]. Figure 2.1 shows a simple representation of a dialogue system. Dialogue systems are generally divided into two categories [49, 112]: (i) *non-task-oriented dialogue systems*, and (ii) *task-oriented dialogue systems*. In what follows, we define each of these categories.

Figure 2.1: Simple representation of a dialogue system.

2.1.1.1 Non-task-oriented dialogue systems

Non-task-oriented dialogue systems are chit-chat systems that focus on conversing with a human on open domains [49, 118]. They are designed for extended conversations to mimic the unstructured conversations or "chats" characteristic of informal human-human interactions [49, 108]. They do not have a predefined goal for the conversation. These systems are mainly designed for entertainment, but also for making *task-oriented dialogue systems* more natural [29, 242]. For example, a study conducted by Bickmore and Cassell [29] showed that people were much more motivated to buy real estate through a task-oriented dialogue system that integrates some chit-chat about topics unrelated to real estate than through a dialogue system that only engaged in task-oriented conversations.

The first chatbot, called Eliza [216], emerged in 1966 from MIT (Massachusetts Institute of Technology). Eliza used a rule-based method where pre-defined outputs were given based on identified patterns in the user utterance. This chatbot was created to simulate psychotherapists[1] by discussing with patients and reformulating most of their statements into questions as shown in Figure 2.2. Recent chatbots include Replika [13] and Mitsuku [11], which are designed to provide emotional support to users through natural language interactions. A more recent chatbot, called ChatGPT [3], emerged in 2022 from OpenAI company [12]. This ChatGPT is designed to assist users with a wide variety of tasks, such as providing information, helping with making decisions, or answering questions on a wide range of subjects, from general knowledge to specific programming concepts. All these recent chatbots have been developed with advanced

[1] a person who treats mental conditions by verbal interaction.

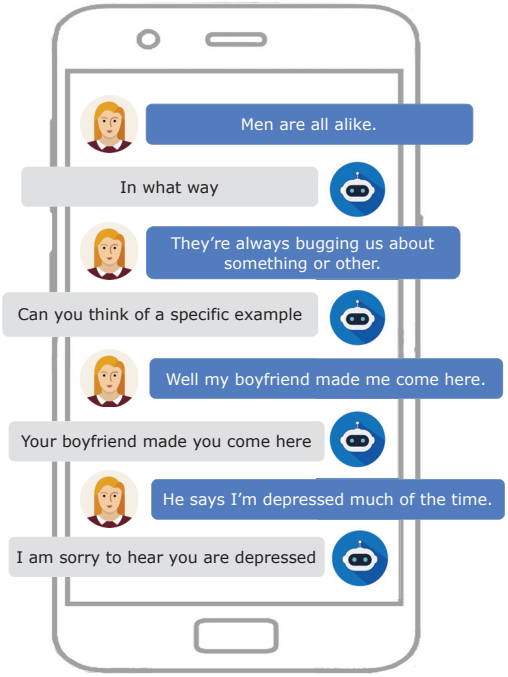

Figure 2.2: Sample Eliza dialogue from Weizenbaum (1966) [112].

machine learning (ML) techniques and have the ability to handle more complex interactions, making the conversation more natural. In general, three major approaches have been developed for non-task-oriented dialogue systems: (1) rule-based methods, (2) generative methods, and (3) retrieval-based methods.

1. Rule-based methods: These methods use a set of pre-defined rules to determine how the chatbot should respond to a given utterance. The rules consist of defining a set of *pattern* and *response* pairs. When a user utterance matches an existing pattern, the chatbot sends the corresponding response [23, 46, 195, 216].

2. Generative methods: These methods use machine learning techniques to generate proper responses during the conversation. The chatbot is trained on a corpus, and the goal is to generate responses that could have never appeared in the corpus [76, 89, 207]. An example of these methods is sequence-to-sequence models [150, 167], which generate an entire response word by word during the conversation.

3. Retrieval-based methods: These methods learn to retrieve information from repos-

itories (e.g., datasets of human-human conversations). They use selection algorithms to select a proper response for the current conversation from a repository [104, 230, 234].

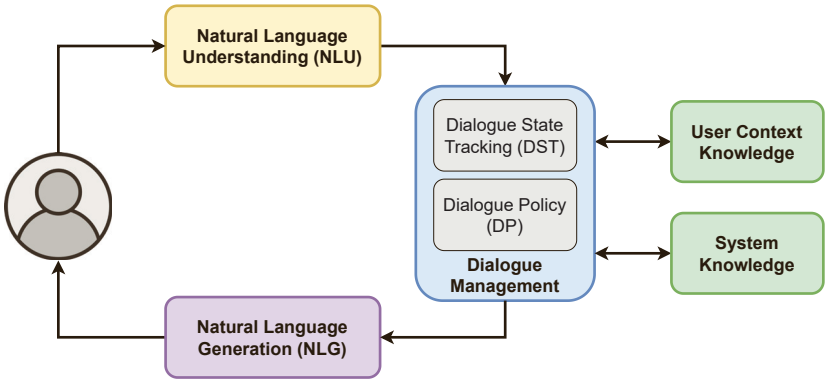

Figure 2.3: Common components of task-oriented chatbot.

2.1.1.2 Task-oriented dialogue systems

Unlike non-task oriented dialogue systems, *task-oriented dialogue systems*[2] aim to complete specific tasks for users (e.g., send a message, schedule a meeting) based on the information provided during the user-bot conversation. These chatbots allow users to interact with software-enabled services in natural language to accomplish users' goals.

A number of techniques have been proposed to build task-oriented chatbots, including rule-based [23] and probabilistic models [100]. Main platforms such as Chatfuel [2] and FlowXO [6] provide flow-based solutions to develop chatbots with zero coding using user interface (UI) elements. Other platforms such as DialogFlow [5], Wit.ai [17], Rasa [169], Amazon Lex [1] and IBM Watson Platform [10], on the other hand, provide machine learning based solutions. In addition to these solutions, a variety of machine learning models have emerged in research following two common architectures: *pipeline* and *end-to-end*. *End-to-end* models, including end-to-end memory networks [253] and sequence-to-sequence models [150], read directly from a user utterance and produce a

[2]Since this thesis focuses on task-oriented dialogue systems, in the rest of the manuscript, the term "chatbot" refers to this type of dialogue system for simplicity.

response. A *pipeline-based* model, on the other hand, is built with a set of components, each responsible for a specific task [51, 174, 198]. Figure 2.3 shows these components together with their interaction flow according to a pipeline architecture [50]. This architecture has been widely adopted by most traditional dialogue systems and still used underlying modern commercial and research systems [77].

In a pipeline architecture, the task-oriented chatbot first uses the *Natural Language Understanding (NLU)* component to convert the user utterance into a structured representation that can be used by chatbot [119]. We will detail this component in Section 2.1.2. Then, the information of this structured representation is fed into the second component, which is the *Dialogue Management* component. The dialogue management component aims to control the conversation flow, to check user inputs, and to choose next actions to perform [34]. A typical dialogue management component has two subcomponents: the *Dialogue State Tracking (DST)* and the *Dialogue Policy (DP)* [34,49]. The DST decides how to infer missing information and maps a given structured representation into a suitable *dialogue state* [227]. Based on the dialogue state, the DP chooses the next action to perform (e.g., invoking an API method) [255]. In Section 2.1.3, we will discuss dialogue management component and in Section 2.2, we will unfold existing techniques and approaches for DST and DP components. The last component, the *Natural Language Generation (NLG)* component, aims to generate a human-like response based on the outcome of the dialogue management component [119,184]. Later, in Section 2.1.4, we will overview NLG techniques.

2.1.2 Natural Language Understanding

The Natural Language Understanding (NLU) component has two fundamental concepts, which are *intent recognition* and *entity recognition*.

User Intents. One of the main objectives of chatbots is to respond to *user intents*. An *intent* refers to the user's goal or purpose. In general, intents are specified using short names including a verb and a noun such as *"schedule-meeting"*, *"book-restaurant"*. Intents are defined by bot developers before starting conversations with users. Then,

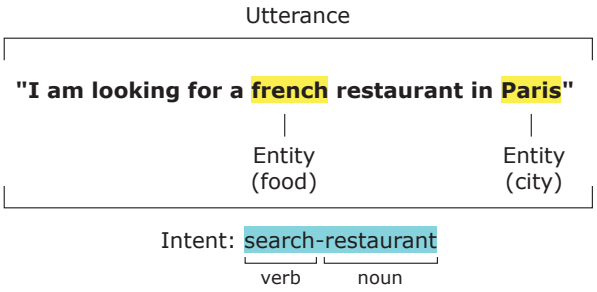

Figure 2.4: An utterance belongs to an intent and contains entities.

during the conversation, the chatbot has to recognize the intent based on the user's utterance. For example, the user utterance *"Please, remind @Boualem that we have meeting tomorrow by 10AM"* corresponds to the intent *"send-reminder"*. There are often several different ways of expressing the same intent. For example, the two utterances *"I am looking for a restaurant"* and *"is there any place to eat"* correspond to the same intent, which is *"search-restaurant"*. Thus, increasing the quality, variety, and quantity of utterances can help to improve intent recognition.

Entities / Slots. In order to fulfill user intent, chatbots often need to extract important information from the user's utterances. Such information, known as *entities* or *slots*, is used to provide a relevant response. For example, in the utterance *"I am looking for a french restaurant in Paris"*, the term *"Paris"* is an entity of type *city*. Figure 2.4 shows an example that illustrates the relationship between utterances, intents, and entities. An utterance belongs to an intent and contains entities.

There are several approaches for recognizing user intents and extracting entities. In the following sections, we will discuss existing approaches for NLU component.

2.1.2.1 Rule-based approaches

Rule-based approaches involve defining a set of handcrafted rules in the form of pattern/response pairs as shown in Figure 2.5. These rules perform NLU, dialogue management, and NLG tasks at once by taking as input the user utterance and producing the corresponding response.

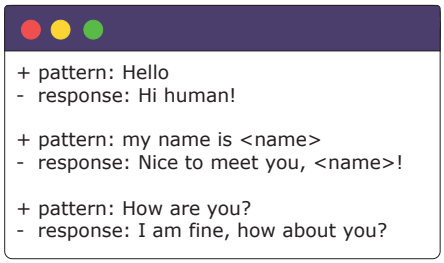

Figure 2.5: Example of Rivescript code.

The set of rules is fully specified by developers or domain experts by using various languages. The first language that was used to develop chatbots is the Artificial Intelligence Markup Language (AIML) [211]. It is based in XML (Extensible Markup Language), and it is open-source. Eliza [217] and its successor Alice [211] are the first chatbots that leverage AIML language. This language is built using two core units: *categories* and *topics*. *Categories* are blocks of rules consisting of a (i) pattern defining user input (i.e., user utterance), and (ii) template defining chatbot response. On the other hand, *topics* are collections of categories. Figure 2.6 shows an example of rules written by AIML. Other rule-based languages include Rivescript [14] and Chatscript [4]. In RiveScript's syntax, the symbol "+" indicates a user input, while "-" denotes the chatbot response. This language provide clearer structure and easier to follow syntax compared to AIML as shown in Figure 2.6.

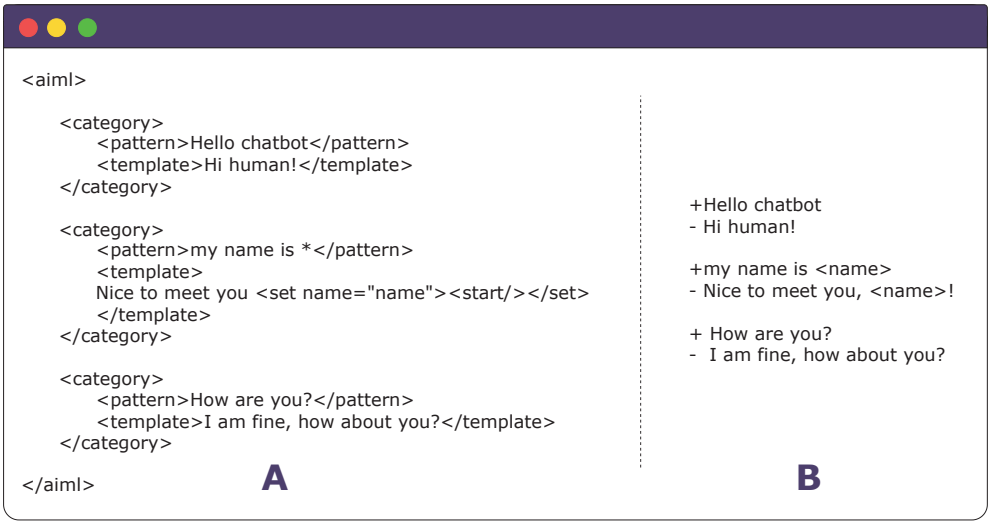

Figure 2.6: Rules written by AIML (A) and Rivescript (B) scripting languages.

Overall, rule-based approach can be a good choice for specific use cases where the scope of the conversation is small and unlikely to grow. An important advantage of a rule-based approach is that it is easy to implement and does not require any training data. However, this approach is not robust to variations in the way people express themselves, thus it lacks flexibility. Additionally, the rule-based approach requires considerable effort from developers to encode rules. As the number of rules grows, finding overlaps and conflicts between rules causes a laborious maintenance cost.

2.1.2.2 Machine learning-based approaches

Machine learning-based approaches can be classified into two categories: (i) classification-based approaches and (ii) deep learning-based approaches.

Classification-based approaches. These approaches consider intent recognition as a classification problem. They aim to analyze the user utterance and to determine which pre-defined intent it belongs to. They involve training a machine learning model on a large dataset of user utterances, each annotated with the corresponding intent and entities (as shown in Figure 2.7). The model is then used to predict the intent and extract entities of new user utterances. Different classification algorithms can be used for recognizing user intents, such as Naive Bayes, MaxEntropy and Support Vector Machine (SVM) [187]. For instance, RasaNLU [169], a natural language processing service, uses SVM algorithm to recognize intent from user utterances.

Deep learning approaches. Deep learning has been utilized effectively for intent recognition due to its consideration of long dependencies between words in utterances [64, 206, 237]. These deep learning approaches convert utterances (from training dataset) into a sequence of numbers where each number refers to an indexed word in the *vocabulary dictionary*. This vocabulary dictionary is built from the training dataset to store unique words. Then, the sequence of numbers is used by a neural network to learn about the sequence pattern of words for each intent. Different variations of neural networks were used to recognize intents and extract entities. For example, works like [95,106,191] applied convolutional neural networks (CNN) to automate features ex-

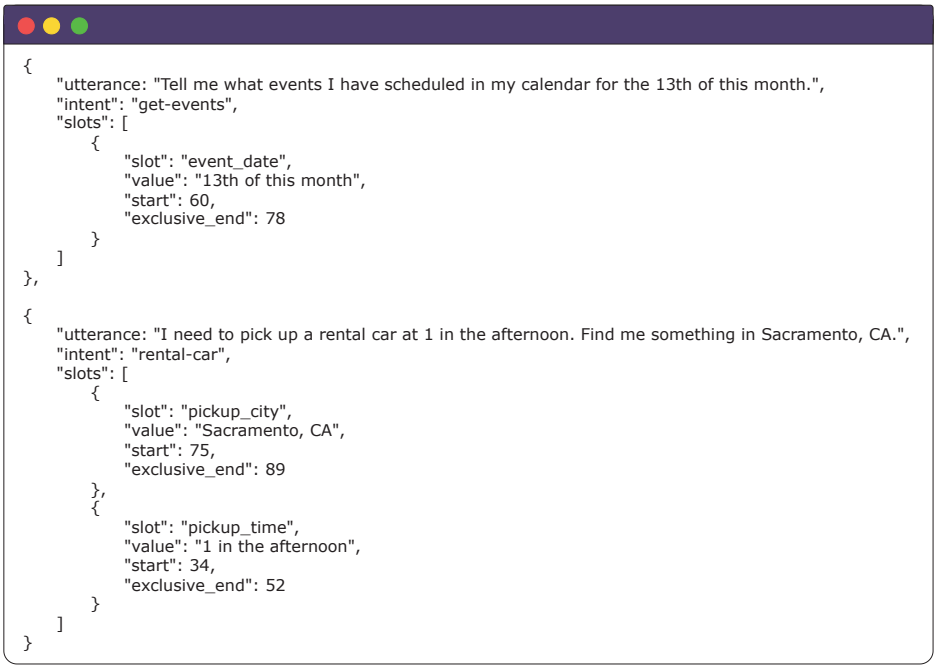

Figure 2.7: An excerpt of training dataset for classification-based intent recognition models.

traction for query classification. The work [171] used recurrent neural network (RNN) approach for intent and entity extraction tasks. In [91], authors proposed to apply recursive neural networks (RecNN) for training of intent recognition task.

Entity recognition is a task in natural language understanding that involves labeling specific information, or *slot* , within an utterance. It is considered as a challenging problem and is often approached as a sequence labeling problem, where a semantic label is assigned to each word [49]. The goal is to take an utterance as input and produce a sequence of slots, one for each word. Researchers have used various methods to tackle this problem, including deep belief networks (DBNs) [64, 65] and recurrent neural networks (RNNs) [156, 185, 239, 240]. Named Entity Recognition (NER) models are also used to extract entity values from sentences [130]. These models typically use a combination of natural language processing techniques, including tokenization, part-of-speech tagging, and chunking to identify named entities in text such as person names and cities. Many different libraries and natural language processing tools have pretrained NER models that can be imported, used and modified according to requirements [190].

Feature Type	Words	Examples
Days	Monday, Tuesday, etc.	Let's do it on Monday.
Ordinal words	first, second, etc.	Only the first option.
Logical words	and, or, etc.	Monday and Tuesday.
Quantifiers	all, any, none, etc.	All options work.

Figure 2.8: Feature categories and examples used to define rules for entity extraction [59].

2.1.2.3 Hybrid approaches

These approaches combines both rule-based and machine learning-based methods for natural language understanding task. For example, Dialogflow service [5] simultaneously uses both of rule-based grammar matching and machine learning matching algorithms to recognize the intent by choosing the best result. The work [59] proposed a chatbot to schedule meetings that utilize predefined rules (shown in Figure 2.8) in addition to classification models to extract relevant information from emails such as meeting duration or time options. This approach exploits both NER models and a set of rules to extract entities from emails (e.g. date, meeting duration). In addition, the work [59] uses crowdworkers to help the chatbot to classify the intent if the model makes a mistake. In other words, if the chatbot is not able to classify an utterance into the corresponding intent, a worker is asked to provide classification given the utterance. Then, the provided class (from the worker) and the given utterance are used to re-train the classification model.

2.1.3 Dialogue Management

Typically all existing chatbots are able to perform simple tasks that can be completed with just one response, with no need for any further questions or exchanges. In other words, these tasks are completed within a *single-turn* style conversation. A *turn* in a conversation is marked by one back-and-forth interaction as shown in Figure 2.9. This means that the user utterance carries all required information (i.e., slots values) to fulfill the user intent (e.g., *"Send a text to Sophia, tell her that I've just left so I'll be there in 10 minutes."*). However, studies on human-bot dialogue patterns [107] reveal that conversations are *multi-turn*, which means that there may exist missing

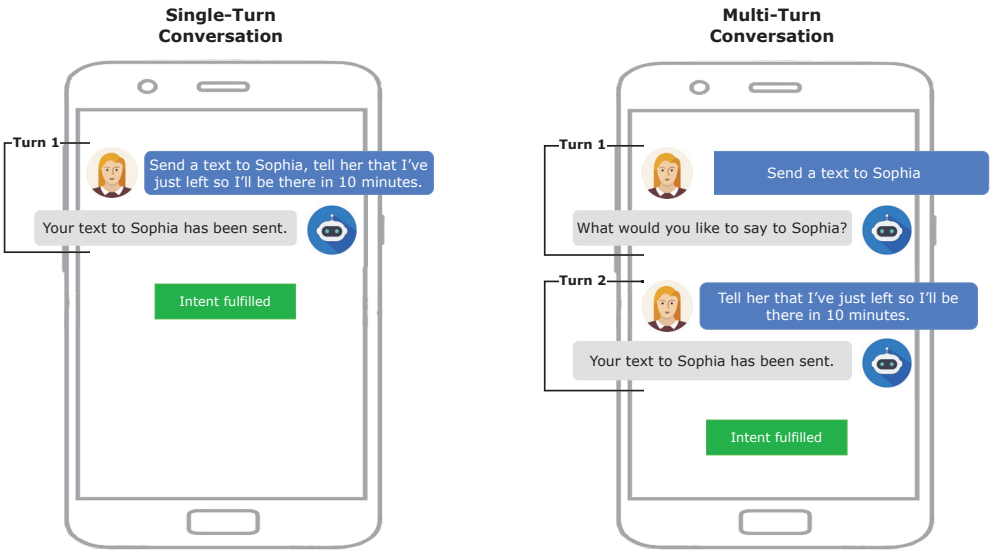

Figure 2.9: Example of single-turn and multi-turn conversations.

information (e.g., text) in user's initial utterance (e.g., *"Send a text to Sophia"*) that needs to be inferred by the chatbot in order to fulfill the user intent. Figure 2.9 illustrates an example of *single-turn* and *multi-turn* conversations. To support multi-turn conversations, the component *dialogue management* has been embedded within chatbots. This component aims to keep track of conversation information that is entered explicitly or implicitly and manages complex interactions with users [34,171]. In Section 2.2, we discuss main approaches that have been adopted to implement the dialogue management models.

2.1.4 Natural Language Generation

Natural Language Generation (NLG) is another component in task-oriented chatbot architecture (Figure 2.3). This component aims to generate a human-like response based on the outcome of the dialogue management component [119, 184]. Two main approaches are used to implement the NLG: (i) template-based approaches, and (ii) generative approaches. In what follows, we discuss each of these approaches.

2.1.4.1 Template-based approach

In the template-based approach, the domain experts define a set of templates, then these pre-written templates are used to generate natural language responses [80, 197, 210]. These responses are generated by replacing placeholders in the templates with data returned by the dialogue management (e.g., data obtained by invoking the OpenWeather API to obtain weather forecast details). This approach is often used in specific tasks such as *get-weather*, *book-restaurant*, and other such intents where the information to be conveyed is predictable and structured [80]. An example is illustrated in Figure 2.10, where the template includes placeholders for the *city*, *weather condition*, and *temperatures*, which are filled with data to generate the chatbot response.

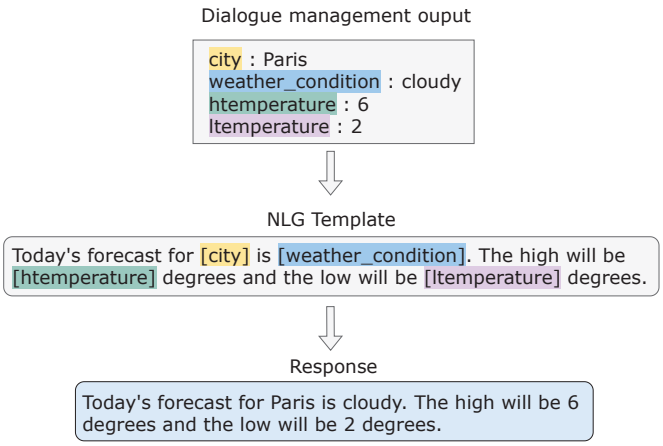

Figure 2.10: Example of a template-based approach.

The template-based approaches are relatively simple to implement for specific domains. However, the main drawback of these approaches is the cost to write and maintain templates, and adapting them to new domains. In addition, these approaches may not be able to handle unexpected input or generate novel responses, as the templates are pre-determined [197, 210].

2.1.4.2 Generative approach

The limitations of template-based approaches have prompted researchers to consider generative approaches to implement NLG component [76, 89, 207]. These approaches

consist of training machine learning models on a large dataset to produce sentences on the fly. During the training, the model learns patterns and relationships between the words and sentences in the dataset [149]. After the training, the model can generate new sentences by sampling from the probability distribution of words it learned during training. For example, given the two sentences *"The cat jumped over the"* and *"The dog chased after the"*, the model can learn that *"cat"* and *"dog"* are nouns that come after *"the"*, and *"jumped"* and *"chased"* are verbs that come before *"over"* and *"after"*, respectively. Thus, given the words [*"under"*, *"the"*, *"ran"*, *"mouse"*], the model can use the patterns it learned during the training phase and generate the new sentence *"The mouse ran under the"*.

Several models are used for generative approach such as neural network to predict the next word given the preceding words in a sequence [27], RNN models based on the Encode-Decoder architecture [56], and LSTM-based model [218] that takes as input a dialogue act (i.e., sentence that serves a function in the dialogue such as *inform* or *ask*) and generates a response based on the given dialogue act.

2.2 Dialogue Management in Task-oriented Chatbots

This section summarize the work on *dialogue management approaches* presented in [34]. Based on the analysis of a wide range of literature, we uncovered three main approaches that have been adopted to implement the dialogue management models (i.e., *Dialogue State Tracking (DST)* and *Dialogue Policy (DP)*). Figure 2.11 shows the three approaches, namely *handcrafted approaches*, *data-driven approaches* and *hybrid approaches*. In what follows, we discuss each one of these approaches.

2.2.1 Handcrafted approaches

Handcrafted approaches rely on programs and models that are fully specified by chatbot developers to track the dialogue state and define the policy (i.e., choose the next action

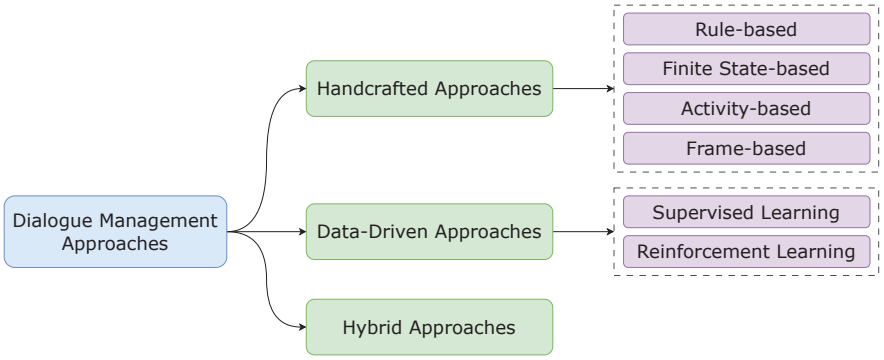

Figure 2.11: Dialogue management approaches.

to do) [113]. We distinguish between four kinds of handcrafted approaches (illustrated in Figure 2.11), namely *rule-based* (discussed in Section 2.1.2.1), *finite state-based*, *activity-based*, and *frame-based* approaches.

2.2.1.1 Finite state-based approach

In finite state-based approach, the user is taken through the dialogue via following a sequence of predetermined states [42,113,154]. Transitions between states are triggered as a result of recognizing a pattern that matches a user utterance. For example, Figure 2.12 represents a dialogue management model defined using a finite state machine (FSM), for a *"flight booking"* scenario. When the chatbot is in the *"One-way trip"* state, it will choose the next state based on the user's answer: (i) *"Yes"* moves to *"Get final confirmation"* state, (ii) *"No"* moves to *"Return date"* state. Notable research chatbots following this approach include Ava [110], Iris [71], Devy [36], Diasy [128] and DialogOS [123].

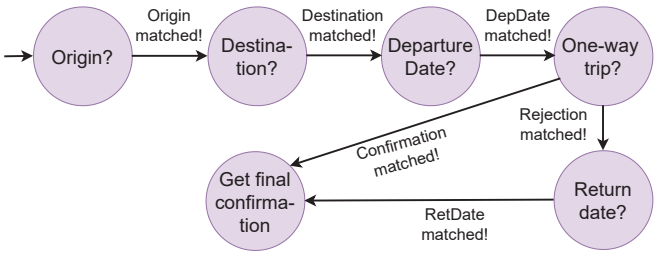

Figure 2.12: Example of dialogue management modeled using a FSM.

The finite state-based approach has the same advantages and limitations as a rule-based. However, it comes with other shortcomings such as versatility and robustness in situations where a user does not follow predefined sequences of states [155].

2.2.1.2 Activity-based approach

The activity-based approach allows the development of dialogue management model by defining *workflows* [193]. While finite state-based approach is a declarative approach that provides a high-level specification of the dialogue management *behavior* in terms of possible states that may occupy during the conversation, the activity-based model is a procedural approach that provides a concrete implementation of dialogue management by precisely specifying the *workflow* that it may go through during the conversation.

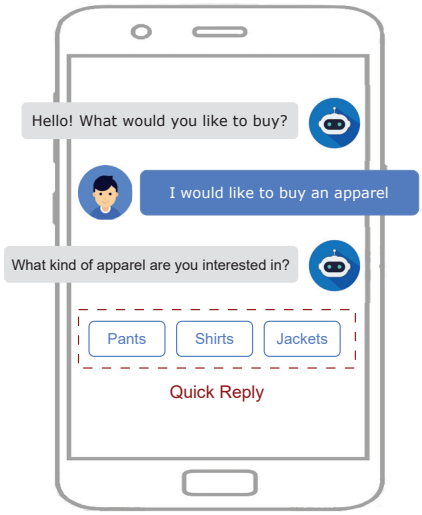

Figure 2.13: Example of a Quick Reply.

There are various platforms that developers can use to build activity-based chatbots such as Chatfuel [2], FlowXO [6], and ManyChat [8]. These platforms provide canvas design to draw conversations using visual elements such as *Quick Reply*. Quick replies are short instant messages that contain possible user responses in the form of buttons as shown in Figure 2.13. The workflow, which is defined by drawing elements in the canvas, is then converted into a list of rules (e.g., Rivescript rules) to deploy a chatbot.

2.2.1.3 Frame-based approach

This approach is based on the concept of frames, which are knowledge structures presenting the different types of information that the chatbot is designed to acquire from the user in order to fulfill a given task (e.g., book travel) [113]. A predefined frame may include a dialogue domain (e.g., travel), an intent (e.g., book travel), and a collection of required slots (e.g., destination, departure date). The goal of a frame-based chatbot is to fill all required slots in its frame with accurate values. The chatbot asks questions to the user until it can fill all slots needed to perform the desired task. It might attach rules to slots to reduce monotony (e.g., if a user has specified a flight destination city, the chatbot can automatically fill the hotel destination slot with the same value). Once all required slots values of a frame are collected, the dialogue management reports back the results of the action associated with the frame to the NLG. This latter relies on a template-based generation to produce the final answer to the user [113].

Compared to finite state-based and activity-based approaches, the frame-based offers more flexibility thanks to its ability to efficiently process over-informative inputs from users while allowing them to fill in the slots in different orders and different combinations. However, considerable testing efforts are needed to ensure that the chatbot would not ask an inappropriate question under any conditions unforeseen at design time. Despite that, the frame-based approach is still until now underlying modern chatbots such as Apple Siri, Amazon Alexa, and the Google Assistant [113].

2.2.2 Data-driven approaches

In contrast to handcrafted approaches where the dialogue management logic has to be defined by hand, data-driven approaches were proposed to learn the dialogue state and the next action to do from data. They involve mainly machine learning approaches, including *supervised learning* and *reinforcement learning* as shown in Figure 2.11. In what follows, we discuss each one of these approaches.

2.2.2.1 Supervised learning approaches

The supervised learning approach learns from a corpus of labeled data, then choose the next action to do depending on what the model learned. A variety of supervised learning models have recently been applied to the dialogue state tracking (DST), dialogue policy (DP), or both as a single module exposed either independently from NLU and NLG components or jointly leading to the emergence of so-called *end-to-end chatbots* [49,253]. In what follows, we first discuss supervised learning approaches for DST and then we move on to those for DP.

Supervised learning based DST. Earliest supervised learning approaches to DST rely on statistical models, including conditional random fields (CRF) [125, 177] and maximum entropy models [157,220,221]. These approaches heavily rely on handcrafted features to learn dialogue state representations. More recently, research has been relying on neural-based approaches, which started to receive more attention, especially with the adoption of deep learning models that have significantly contributed to dialogue management performance improvement. Most of these approaches consider merging NLU and DST into a single model that acts directly on user utterances to update the dialogue state. Notable deep learning models that were initially adopted for DST include multi-layer perceptrons (MLP) [98], recurrent neural networks (RNN) [99,159], and convolutional neural networks [158]. They are used to learn feature representations for user utterances and the associated slot-value pairs. More recent deep learning models [54, 260] adopted graph neural networks (GNN) with attention mechanisms to incorporate slot relations (e.g., similarity) in DST, as a solution to alleviate the data sparsity problem.

We can classify the above DST approaches into two categories: (i) predefined ontology-based where a predefined ontology is provided in advance to define all slots and their values in each domain [54, 158], and (ii) open-vocabulary candidate-generation that estimate the slot value candidates from conversation history and/or language understanding outputs, without any predefined ontology [84, 226, 260]. While predefined ontology-based approaches have been shown to be accurate as they reason over a known

candidate set of each slot, their applicability in practice is still threatened and depends on the ontology coverage. Open-vocabulary approaches, on the other hand, provide a key step toward a DST with zero-shot generalization, whereby adding new intents, slots or even domains do not require the need for collecting new data.

Supervised learning based DP. There are two approaches to applying supervised learning for the DP. The first one considers DP component as a pipelined module, trained independently of DST and NLU modules. In this approach, the DP takes as input the dialogue state to output the next chatbot action. The most widely used supervised learning models to implement a DP are neural networks, bidirectional long short-term memory (BLSTM), convolutional neural networks (CNN) or a combination of the two above (i.e., BLSTM and CNN) [153,198,219]. For instance, in [198], authors represented DP as a neural network with one hidden layer and an output layer consisting of two softmax partitions and six sigmoid partitions (refer to Figure 2.14). For the softmax outputs, one is for predicting dialogue acts (i.e., request, offer, confirm, select), and the other for predicting the associated slots (e.g., food, pricerange, area, none). The sigmoid partitions are used to determine a binary prediction for *offer slots* (i.e., slots the chatbot can mention, such as area, phone number and postcode).

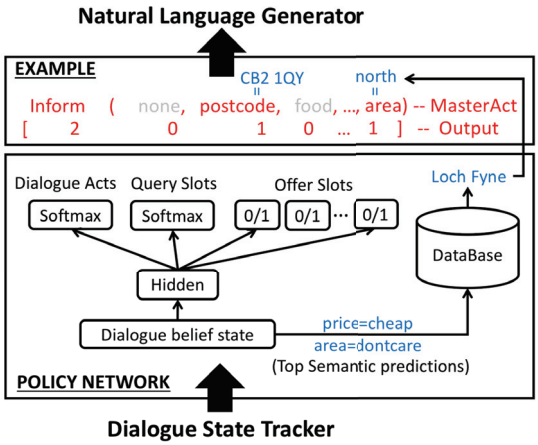

Figure 2.14: Dialogue Policy Network [198].

The second approach is to implement a DP as an end-to-end model that reads directly from a user utterance and produces a chabot action. The sequence-to-sequence model is the main model used in this approach. Such a model is based on the encoder-decoder

architecture which takes a sequence as input and generates another sequence as output. In a chatbot, the source sequence is a user utterance along with a dialogue history, and the target sequence is a corresponding action (e.g., API call, database query).

2.2.2.2 Reinforcement learning approaches

The reinforcement learning (RL) approach treats dialogue management as an optimization problem. It focuses on optimizing the learning over time through experiencing with users by using a series of rewards or punishments and then the chatbot chooses the next action that has the higher reward. Typically, the reward is defined using a function that captures a set of dimensions, such as accomplishment of the task, user satisfaction, dialogue duration. Dialogue management is generally modeled as Markov Decision Process (MDP) [45, 77, 154], where there is a set of dialogue states interconnected with transition probabilities. Another important approach to model RL-based dialogue management is the Partially Observable Markov Decision Process (POMDP). Compared to MDP that relies on the assumption that the dialogue state is fully observable (i.e., is always known to the chatbot with certainty), POMDP caters for unobserved dialogue states [101,138,243]. This makes it able to deal with uncertainties in user utterances [138, 225]. More particularly, the dialogue state is defined as a distribution over all possible states, including the wrong ones arising from an incorrect interpretation or misunderstanding of user utterances.

Recently, research works [52, 61, 72, 137] have applied deep learning models to enhance the performance of RL methods. This leads to a new approach known as *deep reinforcement learning* (DRL). These improvements, however, made RL methods only able to support simple conversations because they basically operate in flat state-action space, hence they have been known as flat RL methods. According to numerous studies [40, 165], flat RL methods are not able to learn well and be data-efficient in large domains and especially where the conversation tasks are complex. These challenges motivate the study of the so-called *Hierarchical RL* [24] which is now being actively explored in order to avoid the curse of state-action space. HRL provides a principled

approach for learning dialogue policies over complex conversation tasks by decomposing complicated conversation tasks (e.g., travel planning) into a sequence of sub-tasks (e.g., book-flight, book-hotel). For example, in [165], authors proposed a dialogue management that has two layers: a top-level layer selects which subtasks to complete, and a low-level layer chooses primitive actions to execute the selected subtask.

Data-driven approaches, including supervised leaning and reinforcement learning based models, contribute to reduce the development and maintenance cost of dialogue management by automatically learning the dialogue state and policies [154]. However, these approaches heavily rely on the quantity and quality of data used for training models.

2.2.3 Hybrid approaches

Hybrid approaches combine either handcrafted or/and data-driven approaches in order to capitalize on the benefits of each. In what follows we discuss (i) hybrid-based DST approaches and (ii) hybrid-based DP approaches. Then, we discuss the advantages and limits of these approaches.

Hybrid-based DST. Hybrid-based DST approaches can be summarized into two categories: (i) combining a rule-based model with a supervised learning model, or (ii) combining multiple supervised learning models. The first approach relies on applying a rule-based model in parallel with a supervised learning model and taking the outputs union of both as a final dialogue state [199, 208, 209]. The second hybrid-based DST category combines the benefits of both using *predefined ontology-based* methods and *open-vocabulary* methods that either dynamically generate a candidate set of slot values or pointing them directly from input utterances [78, 84, 126, 212, 232, 252]. The aim is to allow DST over unknown slot values that are not defined in a domain ontology. In doing this, it seeks to mitigate the scalability issue suffered from the ontology-based method and the low performance of the open-vocabulary method in certain cases.

Hybrid-based DP. There are two methods commonly adopted to learn the dialogue policy using a hybrid approach. The first method consists of integrating a supervised

learning model with handcrafted domain knowledge and rules. The motivation behind this is that some simple operations like sorting a list of database results would be better implemented in a few lines than using thousands of dialogues to learn them [223]. The second method relies on applying multiple DP models simultaneously, whereby the next system action is decided by the policy model that predicts it with the highest confidence [170].

Overall, hybrid-based dialogue management approaches can be considered as an important step to improve the performance of dialogue management and increase its capability to generalize. It has been proven that they can reach performances comparable to purely machine learning models with less training data. While such a hybrid approach may require an amount of developer effort, it can be seen as very useful in practical settings where collecting realistic dialogues for a new domain can be expensive. Approaches combining multiple ML/rule models can provide a potentially stronger dialogue management solution. However, applying multiple ML models at the same time may amplify the need for training data and powerful resource settings.

2.3 Context Knowledge in Task-oriented Chatbots

The work presented in this thesis focuses on the *dialogue state tracking (DST)* component, which is a crucial aspect of task-oriented chatbots. DST is responsible for understanding the user's intent and extracting specific information (referred to as *"slots"*) required to complete a task. In order to accomplish this, chatbots rely on a variety of external knowledge sources, including user context knowledge, crowdsourced information, domain knowledge, and knowledge graphs.

There are two main types of knowledge used in task-oriented chatbots for state tracking: (i) *User Context Knowledge*, and (ii) *System Knowledge* as shown in task-oriented chatbot architecture (Figure 2.3). *User Context Knowledge* refers to any information extracted from the user side and leveraged by the chatbot to infer the dialogue state, such as conversation context and user profile. *System Knowledge*, on the other hand,

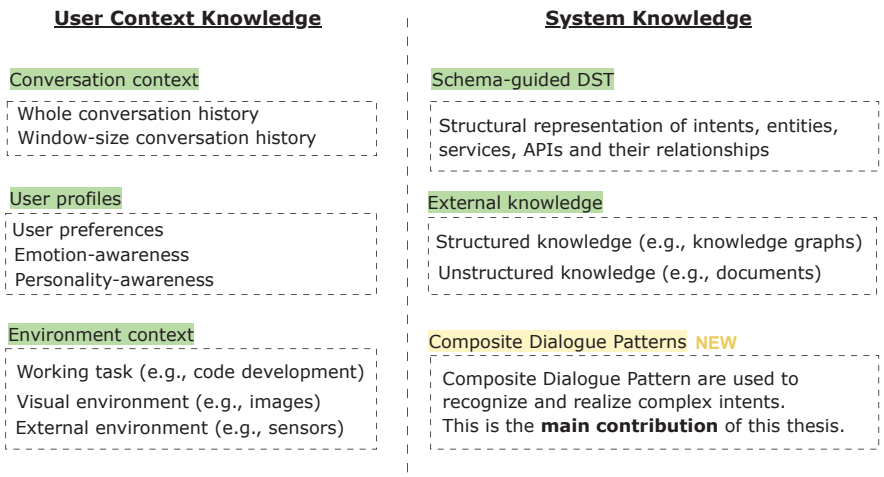

Figure 2.15: Dialogue State Tracking Knowledge.

consists of the knowledge leveraged from the system side to infer the dialogue state. This includes entities and relationships extracted from structured and unstructured data sources, and knowledge graphs, as well as external knowledge such as domain-specific or commonsense knowledge used to complement user context knowledge.

In order to effectively track the dialogue state, the DST component must be able to combine and utilize both *User Context Knowledge* and *System Knowledge*. Figure 2.15 shows main user context and system knowledge types. In what follows, we discuss these types of knowledge.

2.3.1 User Context Knowledge

User context knowledge can be defined as any information extracted from the *user side* and leveraged to fulfill user intent. Existing context knowledge types mainly include *conversation context*, *user profiles*, and *environment context* as shown in Figure 2.15. In what follows, we present each one of these context knowledge types.

2.3.1.1 Conversation context

Also known as *conversation history*, the conversation context includes information that was discussed in the past conversation turns. This conversation context consists of the *whole* or *window-size* of the dialogue history to predict the dialogue state. Deep learning models including HRNN (Hierarchical Recurrent Neural Network) [85], LSTM (Long-Short Term Memory) [79] and BERT (Bidirectional Encoder Representations from Transformers) [250] are utilized to encode the dialogue history.

Some approaches [47, 96, 120, 160, 260] consider only the previous dialogue states to predict the current state instead of taking the whole history. However, this renders them only tailored in pairwise utterance exchanges. They are not very useful for long-dependency dialogue state tracking [235].

Other approaches [86, 127, 178, 189, 227, 233, 251] leverage on the whole conversation history from the first conversation turn up to the last one. In general, an LSTM model over past conversation turns is commonly used to encode the conversation history because of its ability to model long-term dependencies. Concatenating all conversation turns implies one of the reasons that result in increased computational cost [235].

Addressing issues of both kinds of approaches, recent work [92,235] use *fine-grain* turns instead of the last or the whole conversation turns. For example, granularity can be the number of conversation turns spanning from a certain dialogue state in the conversation to the current one (e.g., the 3 or 4 last conversation turns) [235]. In [235], authors conducted studies to explore how the conversation history at different granularities affects the DST. The work presented in [235] showed that there are significant differences in state tracking at different granularities. They found that the determination of the granularity (i.e., number of turns to take into consideration to infer the dialogue state) is according to the characteristics of the used model and dataset. For instance, models with generative decoding, such as TRADE [227], prefer larger granularity because they require more information to track the dialogue state [235].

More advanced approaches [105, 163, 241] focused first on the learning of slot depen-

dencies from the history, allowing them to infer slot values from similar slots. Other advanced approaches [71, 94, 139, 175, 204] leveraged linguistic patterns that are drawn mainly from human-to-human conversations to handle some complex intents and infer their slots. For example, IRIS [71] draws on dependent questions (i.e., one question depends on the answer to some subsequent questions), and anaphora (i.e., expressions that depend on previous expressions) to allow sequencing of intents. These approaches, however, are not enough to capture complex intents that naturally emerge when conversing with services. For example, while the utterance *"Can you book a table for 2 people at Mirazur restaurant for the next public holiday?"* refers to a complex intent requiring a composition of two actions (i.e., getting the next public holiday, then booking the restaurant), it cannot be recognized by IRIS [71]. The reason is that IRIS can recognize composition based only on linguistic features (e.g., a composition is recognized when a user answers with a new intent to the chatbot question for a missing slot).

2.3.1.2 User profiles

A user profile in chatbots is a data structure where the information related to a given user is stored [182]. The information of a user profile can be *general user attributes*, which include personal information (e.g., gender, age, current language, preferences, hobbies), *general statistics* (e.g., the number of sessions, dialogues and dialogue turns, the date of the last interaction with the chatbot), or *usage statistics* that correspond to actions over the system that a user performs [43, 182, 228]. Typically, personalized chatbot models use *user profiles* to be able to capture users' personal preferences and return personalized responses.

The first benchmark dataset for personalized chatbots was proposed by [111]. They used a Memory-based neural networks model (MemNN) to encode the user profile [111]. They also extended the original MemNN by dividing the memory of the model into two memories: (i) profile memory and (ii) conversation memory. Similarly, [145] introduced a personalized MemNN, which learns distributed embeddings for user profiles and the dialogue history from users with the same gender and age. Likewise, [249] proposed to

incorporate a retrieval module into the MemNN model, which improves the performance by retrieving the relevant responses from other users. The above approaches assume that complete user profiles can be obtained by asking users to fill in all blanks in user profiles, which may be unrealistic in practice.

Other approaches [129, 202, 203] focus on inferring missing information in user profiles (e.g., inferring information from previous conversations), then apply one of the above models. However, these approaches have to train a model to infer missing user profiles before starting the conversation, thus inference errors may happen during the conversation. Recent work [164] proposed a cooperative MemNN, which introduces a cooperative mechanism to enrich user profiles gradually as dialogues progress, which improves response selection based on enriched profiles simultaneously. While the above approaches focus on single-domain tasks, a recent work [214] proposed a generator-reranker framework, using GPT-2 as generator and BERT as reranker, to support personalization across a wide range of tasks in multi-task conversations.

Leveraging user profiles in task-oriented chatbots is an area that has more than what we are discussing in this thesis [73, 90, 140, 147, 196]. For instance, there are *empathetic dialogue systems*, which are proposed to improve the perception and expression of emotional states and personal preferences [147]. Empathy refers to the capability to imagine oneself in the other's situation and to experience the emotions that she/he is experiencing [162]. Emotion-awareness [132, 133] and personality-awareness [129] are two key features that underpin empathetic dialogue systems [147].

2.3.1.3 Environment context

An environment context is related to the task that the user is working on and what the chatbot can get from this environment without asking the user [20, 38, 116, 161, 186].

For example, a chatbot can retrieve information from a *working task environment* [38, 161]. In [38], authors proposed Devy chatbot to provide automated support in DevOps processes. Devy relies on domain-specific services that keep track of information

related to the user task working environment (e.g., code development), including active projects, issues, code reviewers, etc. The main goal is to infer the required slots for performing DevOps actions without developer involvement. For instance, after a developer completes a code, she/he can say *"I'm done"*. Thanks to the information it has been able to keep from the user task environment, Devy is expected to complete automatically the remaining DevOps actions including committing code changes, pushing them to a remote repository, assigning reviewers, etc., without any intervention from the developer side.

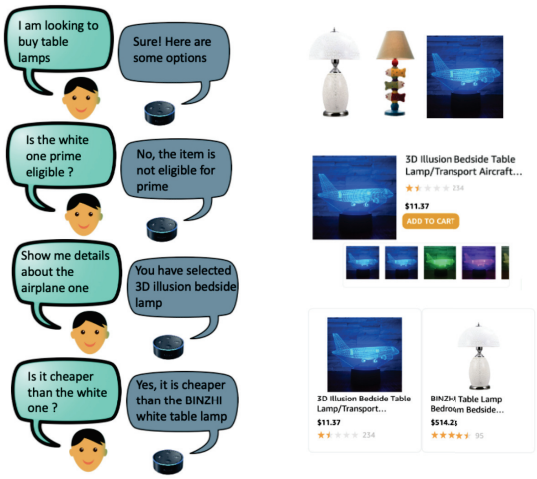

Figure 2.16: Example Multimodal Conversation [20].

Chatbots can also retrieve information from *visual environment* [20, 63, 124, 186, 205]. For example, in [20], authors proposed an approach to support *multimodal conversation* and infer slots values from a shared visual context (e.g. a device screen) and fulfill user intent. For instance, when a user browsing to shop for tables on a speech-enabled smart TV says: *"what is the price of the white round table?"*, the chatbot should be able to identify the right product based on its visual characteristics and then responds with the price [20]. Figure 2.16 shows an example of multimodal conversation. The user can refer to visual entities by attributes such as color (e.g., *"the white one"*) or shape (e.g., *"airplane one"*). She/he can also refer to the visual elements using associated metadata such as prime-eligibility (e.g., *"is it prime eligible?"*).

Another source of environment can be *sensors*. For example, in the work [186] authors

proposed a museum chatbot that utilizes exhibition sensors to guide visitors to different exhibits based on their current location within the museum. When a visitor asks the chatbot *"where does it go on?"*, the exhibition sensor data is used to determine their current location and provide directions to the next exhibit on the tour. This allows the chatbot to provide a more personalized and relevant experience for the visitor by using sensor data to understand and respond to the visitor's current context [186].

2.3.2 System Knowledge

System Knowledge consists of the knowledge leveraged from the *system side* to infer the dialogue state. This include entities and relationships extracted from structured and unstructured data sources, knowledge graphs, and external knowledge such as domain-specific or commonsense knowledge used to complement *user context knowledge*. In what follows, we first discuss end-to-end approaches (Section 2.3.2.1), then we discuss DST approaches that leverage schemas for capturing the structural representation of conversation data to predict the dialogue state (Section 2.3.2.2), finally we discuss DST approaches that use external knowledge in addition to schemas to improve the chatbot understanding (Section 2.3.2.3).

2.3.2.1 End-to-end approaches

Some chatbots use end-to-end systems [102] that rely on user context knowledge and generative models to generate answers (e.g., question and answering services). These systems consider NLU, DST, and DP as a single module. End-to-end models can be considered as "black-boxes" that accept user utterances as input and return new system states/actions as output. The sequence-to-sequence model [150] is the main model used in this approach. Such a model is based on the encoder-decoder architecture which takes a sequence as input and generates another sequence as output. In chatbots, the source sequence is a user utterance along with a dialogue history, and the target sequence is a corresponding action (e.g., API method call). The sequence-to-sequence model is initially implemented using RNN with LSTM cells, where the hidden state of RNN is

utilized as the representation of a dialogue state. This model is later augmented with an attention mechanism to improve its ability to handle long-term dependency [150, 257]. Using end-to-end memory networks is another alternative to build end-to-end models. Compared to the pipelined policy, the key advantage of the end-to-end approach is inferring the representation of the dialogue state which avoids the design of its related features. However, this requires a lot of training data that may be expensive to collect. Thus, schema-guided DST approaches were proposed.

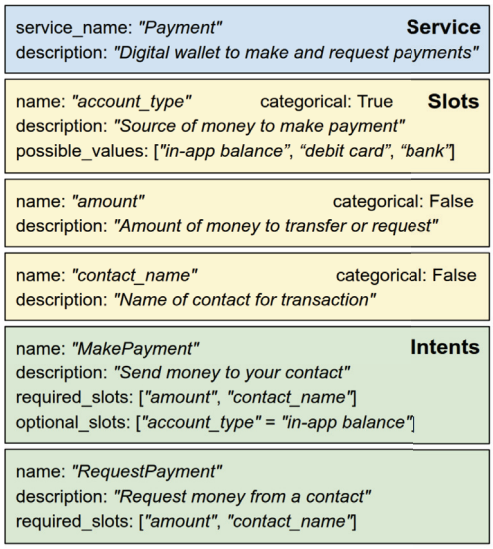

Figure 2.17: Schema example for a wallet service [173].

2.3.2.2 Schema-guided Dialogue State Tracking approaches

Most of task-oriented chatbots leverage schemas capturing the structural representation of conversation data to predict the dialogue state. For instance, the work [173] introduced a unified schema defining a service or API as a combination of intents and slots. Figure 2.17 shows an example of this schema for a digital wallet service. A BERT-based state tracking model then takes this schema as input to enable the recognition of intents and inferring their slot-value pairs. The captured knowledge (i.e., the unified schema), however, can only help in the recognition of basic intents.

Works like [53, 136, 229] use a slot-level schema graph that captures dependencies be-

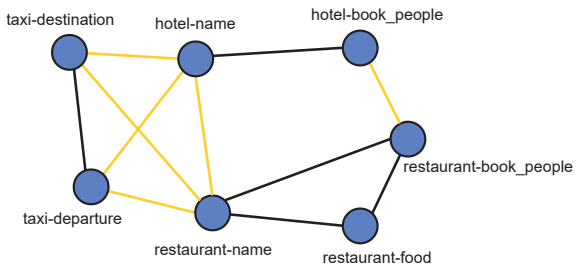

Figure 2.18: Domain specific slot-level schema graph [53].

tween slots. The aim is to allow the state tracking model to infer slot values from similar slots. For instance, Chen et al. [53] proposed two step-based state tracking model that first relied on a graph attention matching network (GAMT) to learn contextual features for each slot by fusing information from both slot-domain schema graph and utterance. Then it utilises a recurrent graph attention network that takes the obtained slot features and a slot schema graph capturing relations amongst all slots, to infer the slot value. An example of domain specific slot-level schema graph is represented in Figure 2.18. The nodes of this schema graph consist of all slots (e.g., *hotel-name*, *taxi-destination*). There is an edge between slots that belong to the same domain, and slots that may share the same values. For example, in Figure 2.18, there is an edge between *taxi-destination* and *restaurant-name* because a *taxi-destination* can be a *restaurant-name*.

Other efforts like [135] leverage the backend database schema in state tracking model to allow slot inference from the database entities. These methods, however, work with chatbots integrated only with databases, where the inferred slot-value pairs are used to frame the query.

To improve the accuracy of state tracking, chatbots use various additional sources of knowledge to reason about potentially ambiguous user intents and match them to underlying services (e.g., databases, APIs) that realize such intents. In the Section 2.3.2.3, we discuss another system knowledge type.

2.3.2.3 Content-based approaches

There are dialogue state tracking approaches that use external knowledge in addition to schemas. External knowledge, including domain-specific or commonsense knowledge, is usually used to complement user context knowledge with additional background [131,147,147]. Retrieving and representing a large-scale knowledge base remains challenging [131]. This knowledge can be classified into two main categories based on if the knowledge is structured or not.

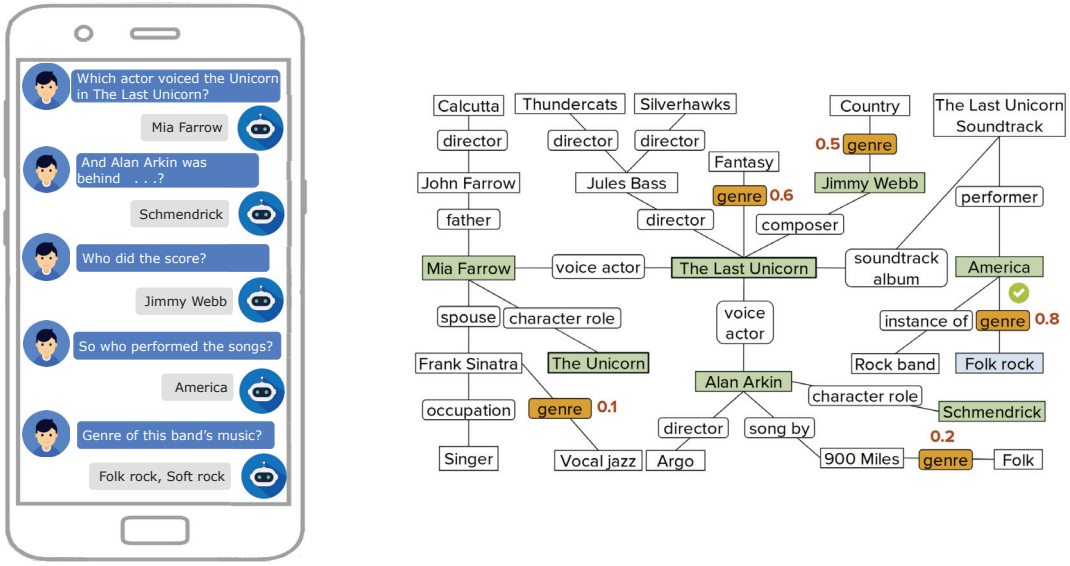

Figure 2.19: Example of fact-centric questions conversation and its corresponding knowledge subgraph [57].

Structured knowledge allows a chatbot to represent and retrieve relational information about different kinds of entities [147]. Examples of structured knowledge include databases and knowledge graphs (KG) [57,58,136,229]. In this context, efforts like [57] proposed an approach, called CONVEX (CONVersational KG-QA with context EXpansion), to handle *incomplete fact-centric questions* in a conversation. Figure 2.19 illustrates an example of such conversation, which is typically characterized by a complete initial fact-centric question with an incomplete follow-ups, an initial and often central entity of interest (e.g., *"The Last Unicorn"*), and slight shifts in focus (inquiry of the band America's genre). The proposed system answers such questions by using a KG and maintaining context using entities and predicates seen in previous questions.

Authors [57] use the initial question to identify a small subgraph of the KG for retrieving answers (example of such subgraph is represented in Figure 2.19). Then, they dynamically expend the context in the form of a subgraph as the conversation proceeds. The core of the approach is a graph exploration algorithm that expands a frontier to find candidate answers for the current question.

Some other work [67,87,131,143,259] focused on unstructured knowledge, such as external documents and online reviews, to retrieve more knowledge. Since this unstructured knowledge is usually in the form of plain texts, sequence encoders [83] are mostly used to encode this knowledge.

2.4 Summary and Discussion

In this section, we summarize the state of the art, then we discuss some research issues.

2.4.1 Summary

In this section, we summarize dialogue management approaches (Table 2.1), knowledge sources in task-oriented chatbots (Table 2.2), and the most well-known chatbot development tools (Table 2.3).

2.4.1.1 Summary: Dialogue Management in Task-oriented Chatbots

As shown in Table 2.1, dialogue management approaches can be classified into three categories: handcrafted approaches, data-driven approaches, and hybrid approaches. In what follows, we provide a summary of strengths and weaknesses of these approaches.

- **Handcrafted approaches** offer a simple way to design dialogue management which is helpful for quickly creating chatbots. They work particularly well in

Table 2.1: Summary of strengths and weaknesses of dialogue management approaches.

Methods	Sub-Methods		Strengths	Weaknesses
Handcrafted	**All Methods**		- Easy to implement when the conversation scope is small. - Do not require any training data. - Enable the conversation traceability.	- Require high development and maintenance cost. - Have limited scalability and robustness. - Do not consider user feedbacks to adapt the DP.
Data-driven	**SL Models**		- Tracking the dialogue state and choosing the next action can be fully automated. - Require less effort to be adapted to new domains. - Able to deal with the natural language variations.	- Heavily rely on the quantity and quality of training data. - Do not consider user feedbacks to adapt the dialogue policy.
	RL Models	**Flat RL**	- Can adapt to different user behaviors. - Able to deal with uncertainty in user utterances. - Feature-engineering of state/action spaces can be alleviated with the adoption of DL models	- Heavily rely on the quantity and quality of training data. - Limited to small-scale conversation domains.
		HRL		- Heavily rely on the quantity and quality of training data. - Requires domain-specific knowledge to specify good task hierarchies.
Hybrid	**Rule and ML**		- Reduce the learning complexity and the amount of training data. - Provide control over conversation flow.	- Suffer from poor domain portability.
	Multiple ML		- Improve performance.	- Rely on the quantity and quality of training data. - Require powerful resource setting.

conversation scenarios with clear structure and objectives. One important advantage of handcrafted methods is the conversation flow traceability which helps to track the interpretation of user utterances and chatbot actions for further fixes/improvements. However, these methods lack flexibility because users cannot express their utterances in any way they desire (i.e., there may not be a rule that corresponds to the user utterance). In addition, these methods require a significant amount of development and maintenance effort due to the need to manually design features such as rules and interaction models.

- **Data-driven approaches** offer a cost-effective solution for developing and maintaining dialogue management. By leveraging deep neural models and word embedding techniques, these approaches can automatically learn policies and dialogue state features, such as slots and slot values, without the need for a predefined ontology. This makes dialogue management more scalable and adaptable to changing dialogue domains, as it can handle the addition of new slots and values not present in the training data. Despite their benefits, existing data-driven approaches are heavily dependent on the quality and quantity of training data. Additionally, they may not provide clear integration of API invocations.

- **Hybrid approaches** involving diverse combinations of handcrafted and data-driven models can be considered as an important step to improve the performance of dialogue management and increase its capability to generalize. In particular, approaches combining ML model with a set of rules grant more flexibility to application developers to control conversation flow and ensure that it is adequately aligned with business rules. In addition, it has been proven that they can reach performances comparable to purely ML models with less training data. While such a hybrid approach may require an amount of developer effort, it can be seen as very useful in practical settings where collecting realistic dialogues for a new domain can be expensive. However, applying multiple ML models at the same time may amplify the need for training data and powerful resource settings.

Table 2.2: Summary of knowledge sources in task-oriented chatbots.

Knowledges sources		Examples	References
User Context Knowledge refers to any information extracted from the user side and leveraged by the chatbot to infer the dialogue state.		**Conversation Context** includes information that was discussed in the past conversation turns.	[30, 36, 40, 41, 47, 60, 61, 71, 72, 82, 84, 86, 88, 92, 94, 96, 98, 99, 105, 120, 125, 127, 128, 137, 139, 157–160, 163, 165, 166, 170, 172, 175, 177, 178, 183, 188, 189, 194, 195, 199, 201, 204, 208, 209, 215, 220–224, 227, 231, 233, 235, 236, 241, 251, 254, 256, 260]
		User Profiles is a data structure where the information related to a given user is stored such as user personal information.	[43, 73, 90, 111, 129, 132, 133, 140, 145, 147, 162, 164, 182, 196, 202, 203, 214, 228, 249]
		Environment Context is related to the task that the user is working on and what the chatbot can get from this environment without asking the user	[20, 38, 63, 116, 161, 186, 205]
System Knowledge consists of the knowledge leveraged from the system side to infer the dialogue state such as entities and their relationships.		**Schema-guided DST approaches** leverage schemas capturing the structural representation of conversation data to predict the dialogue state.	[53, 135, 136, 173, 229]
	Content-based	**Structured knowledge** such as databases and knowledge graphs.	[57, 58, 136, 147, 229]
	Content-based	**Unstructured knowledge** such as external documents and online reviews.	[67, 87, 131, 143, 259]

2.4.1.2 Summary: Context Knowledge in Task-oriented Chatbots

Table 2.2 summarize main knowledge sources in task-oriented chatbots including *user context* and *system* knowledge. In what follows, we provide a summary of each of these knowledge sources.

- **User Context Knowledge** can be defined as any information extracted from the *user side* and leveraged to fulfill user intent. Examples of this knowledge are:
  - **Conversation Context** includes information that was discussed in the past conversation turns. This conversation context consists of the whole or window-size of the dialogue history to predict the dialogue state. We noted that the conversation history has been the widely covered feature in the DST state-of-art approaches. Leveraging the conversation history has proven essential to hold complex and multi-turn conversations, as processing missing information in the dialogue state may require the DST model to go beyond multiple previous turns up to the last turn.
  - **User Profiles** refer to the collection of information related to a given user such as user personal information and preferences. This information is used by the chatbot to personalize the conversation and improve the user experience by providing tailored responses. User profiles are created during the initial conversation and updated over time to reflect changes in the user's preferences. Models such as memory-based neural networks model (MemNN) and cooperative MemNN are used to enrich user profiles gradually as dialogues progress.
  - **Environment Context** is related to the task that the user is working on and what the chatbot can get from this environment without asking the user. For example, a chatbot can retrieve information from a working task environment (e.g., development environment), from a visual environment (e.g., a video), or even from sensors (e.g., getting user location by using sensors such as GPS).

- **System Knowledge** consists of the knowledge leveraged from the *system side* to infer the dialogue state. This include entities and relationships, and external knowledge such as domain-specific or commonsense knowledge used to complement *user context knowledge*. Examples of this knowledge are:
  - **Schema-guided DST approaches** leverage schemas capturing the structural representation of conversation data to predict the dialogue state. Typically, a schema consists of defining different elements (such as domains, intents, slots and services) and relationships between these elements.
  - **Content-based approaches** use external knowledge in addition to schemas. External knowledge, including domain-specific or commonsense knowledge, is usually used to complement user context knowledge with additional background. This knowledge can be classified into two main categories: structured knowledge such as knowledge graphs and databases and unstructured knowledge such as external documents and online reviews. Leveraging external knowledge was the less covered feature in chatbots.

2.4.1.3 Summary: Chatbot development tools

Over the past decade, a number of companies have launched cloud-based NLU tools, with the goal of enabling developers to enhance their existing NLU products or develop new conversational assistants more easily. In this section, based on works [19, 44], we summarize most well-known chatbot development tools as shown in Table 2.3. We identify six main chatbot development platforms: (1) Google's DialogFlow [5], (2) Rasa [169], (3) Facebook's wit.ai [17], (4) Microsoft LUIS [10], (5) IBM Watson Platform [7], and (6) Amazon Lex [1]. In Table 2.3, we compare between these platforms based on 10 features, ranging from usability to pricing:

- *Usability* indicates the perceived ease of use of the platform. From *high* (simple and intuitive for a developer) to *low* (difficult to use).

Table 2.3: Summary of most well-known chatbot development tools [44].

Platform	Usability	Natural Languages	Programming Languages	Pre-build Entities	Pre-build Intents	Default Fallback Intent	Automatic Context	Online Integration	Webhook/ SDK Availability	Price
Dialog Flow	High	25	11, including Java and Ruby	60	34	Yes	Yes	14, including Telegram and Alexa	Webhook and SDKs	Free
Rasa	Medium	17	Any programming language that can run as a web server	15	-	Yes	No	20, including Slack, FB Messenger	Webhook and SDKs	Free
Wit.ai	Medium	132	3 Node.js, Python, and Ruby	22	Zero	Yes	No	Zero	SDK	Free, contact heavy usage
LUIS	Medium	20	4, Android, Python, Node.js, and C#	13	20	Yes	No	Zero	Webhook and SDKs	Free up to 10k requests per month
Watson Platform	High	13	6, including Node.js and Java	7	Zero	Yes	Yes	Zero	SDK	Free up to 10k requests per month
Amazon Lex	Low	7	9, including Java and Go	93	15	Yes	Yes	3, Twilio SMS, FB Messenger, and Slack	SDK	Free for the 1st year (with limits)

- *Natural Languages* indicates how many natural languages the platform supports.
- *Programming Languages* indicates how many programming languages the platform supports.
- *Pre-build Entities* reports how many pre-build entities the NLU tool offers.
- *Pre-build Intents* reports how many pre-build intents the NLU tool offers.
- *Default Fallback Intent* indicates whether the platform has a fallback mechanism for intents which allows the proper classification of utterances that are not recognized as part of existing intents.
- *Automatic Context* indicates whether the platform can automatically manage the context in a conversation.
- *Online Integration* indicates which third-party integrations are available.
- *Webhook/SDK Availability* indicates whether a developer can integrate his/her chatbot with other software.
- *Price* indicates the pricing for using the platform.

The selected platforms utilize machine learning algorithms that are totally transparent for the bot developers. While they share certain functionality such as being cloud-based and supporting multiple programming languages and natural languages, they also differ in other aspects. In what follows, we describe the main characteristics of each platform.

- **Dialogflow** is a NLU cloud platform owned by Google. It provides a free-to-use conversational interface and supports different languages and programming languages. It also provides context management to let developers control conversation flows. Moreover, DialogFlow offers a range of built-in integrations with other chatbot-based platforms, including Telegram, Google Assistant, and Amazon Alexa. In addition, developers can insert answers directly into the web interface or use an ad-hoc server applications through the webhook mechanism enabled by the DialogFlow APIs.

- **Rasa** is an open-source machine learning framework for building chatbots that offers a high degree of flexibility and customizability. Rasa is popular among developers who prefer open-source tools and want full control over the chatbot's behavior and performance. Rasa has a large and active community of contributors and users, and it offers various integrations and plugins to enhance the chatbot's functionality.

- **Wit.ai** is a NLU cloud platform owned by Facebook. It offers support for multiple natural languages but only three programming languages. Unlike other NLU tools, Wit.ai focuses on extracting entities from single sentences, functioning as a NLU parser rather than a complete NLU platform. Wit.ai does not offer any chatbot-based platform integration, web interface for handling conversations, or context management tools. Therefore, developpers must realize any desired integration, conversational aspects, or other features within their own code.

- **LUIS** (**L**anguage **U**nderstanding **I**ntelligent **S**ervice) is the NLU cloud platform of Microsoft, part of the Azure cloud services. Being integrated into Azure, LUIS shares the pricing schema with it and can access some additional features. It supports various languages, but only four SDKs are available. LUIS is based on the active learning technology and offers a set of programmatic REST APIs that can be used to automate the application creation process.

- **Watson Platform** is the NLU cloud platform of IBM, part of IBM Bluemix cloud services. Like LUIS, Waston Assitant shares Bluemix pricing schema and may access to additional features. It supports multiple programming and natural languages. Watson Platform utilizes two contexts (i.e., conversation history and a corpus that is available to it) to gain a degree of confidence in interpreting questions and for finding responses.

- **Amazon Lex** is the NLU platform part of the Amazon Web Services (AWS). It shares the pricing schema with AWS and may access to additional features. It supports various programming languages but only 7 natural languages. It provides the advanced deep learning functionalities of automatic speech recognition

(ASR) for converting speech to text, and NLU functionalities to extract intents and entities from user utterances.

2.4.2 Discussion

Despite the aforementioned achievements, creating scalable and robust dialogue management techniques that can emulate human-like conversations remains a deeply challenging problem. In this section, we will discuss some research issues in this field.

- **Data control and quality.** The success of data-driven and hybrid approaches largely relies on the availability of high-quality training data, which still an open issue. Obtaining high-quality training data requires effective methodologies and processes for selecting, pipelining, tuning, and controlling data acquisition tasks. This represents a key research area that has a significant impact on the performance of dialogue management and subsequently improving the chatbots.

- **Handling conversation breakdowns.** Persistent concerns such as misunderstandings, disagreements, inappropriate responses, complaints, and rejection of offers can lead to breakdowns in conversations, resulting in negative impacts on the user experience. Chatbots are currently incapable of managing these breakdowns due to the diverse nature of user utterances and the ambiguity of natural language. To overcome this, it is crucial to endow dialogue management techniques with efficient strategies that can detect and fix potential conversation breakdowns automatically.

- **Automated generation and formal verification of dialogue management models.** Handcrafted approaches are still the suitable choices for conversations where user inputs can be known a prior and for any application domain where determinism property is required. However, current approaches suffer from high development and maintenance costs. Thus, having automated mechanisms that better leverage existing resource models (such as ontologies, knowledge graphs, business processes) is a key step toward semi or even fully automated generation

of handcrafted dialogue management models. Furthermore, formal verification of such models may still be required to ensure their reliability.

- **Towards explainable conversations**. Explainability is an important aspect to make the conversation more human-like. In this context, the dialogue management must be capable of justifying its actions and decisions. This is beneficial not only for developers to evaluate their models but also for end-users to gain greater transparency and ask questions about a chatbot's decision-making process.

- **Supporting composite user intents.** Developing chatbots capable of handling complex conversations spanning multiple topics and domains is an ongoing area of research. However, building such chatbots presents significant challenges, particularly in the need for rich abstractions to capture complex user intents, and the integration of latent knowledge required to recognize composite intents and translate them into APIs and their compositions. As of now, this knowledge is rarely codified and utilized in chatbot development. In this context, there is a need for more flexible techniques that cater for composite intent recognition.

In this thesis, we focus on addressing the research issue of supporting complex/composite user intents. While research from the previous efforts is certainly complementary and some elements are adopted in our contributions, most of them do not focus on augmenting intents with knowledge that facilitates the superimposition of natural language conversations over process and software-enabled services and use of such knowledge to support dynamic synthesis of services. To the best of our knowledge, the work presented in this manuscript is the first to identify and characterize a set of composite dialogue patterns to recognize and realize different classes of composite intents. We proposed to build upon advances in ML techniques to enable the recognition of basic intents, but contribute a new approach to recognize composite intents. This approach will be detailed in the next chapters.

Chapter 3

Composite Dialogue Patterns

Reusable Abstractions and Patterns for Recognizing compositional conversational flows

Contents

The content of this chapter is an extension of the work presented in [33]. In this work, we identified and characterized a set of composite dialogue patterns and extended a conversational model [248] to represent them. These patterns endow bot platforms

with reusable functionality to recognise compositional conversational flows, that would otherwise have to be implemented by bot developers.

The rest of this chapter is organized as follows: We start with an introduction in Section 3.1. In Section 3.2 we discuss related work. In Section 3.3 we explain how human-chatbot conversations are formulated. In Section 3.4 we present a conversational model that we adopted and extended to represent the identified composite dialogue patterns. In Section 3.5 we characterize these composite dialogue patterns in the adopted model. Section 3.6 presents validation of the proposed patterns, and finally we provide a conclusion in Section 3.7.

3.1 Introduction

Task-oriented chatbots emerged as a paradigm to naturally access services and perform tasks through natural language conversations with software-enabled services and humans [34]. They enable the understanding of user utterances, expressed in natural language, and on fulfilling such needs by invoking the appropriate backend services (e.g., APIs) [34]. Fulfilling a user request consists of: (1) understanding the *user utterance* expressed in natural language (e.g., *"what is the weather in Paris?"*), (2) recognizing the *user intent* corresponding to task(s) that the user wants to accomplish (e.g., `get-weather`), (3) extracting relevant *slot-value* pairs (e.g., `location: Paris`), (4) invoking the corresponding backend service that fulfills the user intent (e.g., call `OpenWeatherMap` API method to get weather condition), and (5) returning a natural language response (e.g., *"we have light rain in Paris"*).

Ideally, a chatbot should detect intents and infer slot values with the least possible interactions with the user (i.e., the chatbot asks the user for a missing value only when it cannot infer it from other sources). A key challenge to achieve this objective is devising robust intent recognition and slot inference despite the potentially ambiguous and complex utterances. An utterance may not always follow a simple conversation pattern, where the chatbot recognizes a *basic intent* and infers all required slot values

from the utterance, as in the previous example.

Natural user conversations can be rich, potentially ambiguous, and express *composite user intents* [71, 245]. In a *basic intent* the chatbot infers all required slot values from the user utterance and calls the corresponding service to fulfill the intent. The realization of a *composite intent*, however, requires the chatbot to break it down into a list of atomic actions and infer potentially missing values from different sources, not directly from the utterance. For example, given the utterance *"Can you book a table for 2 people at Mirazur restaurant for the next public holiday?"*, the chatbot should be able to infer the information such as *number of people* and *restaurant name* from this utterance; however, it also needs to search when will the next holiday be. Also, a human may omit required slot values in an utterance because some of this information can be naturally inferred from other *sources*, such as conversation history, commonsense knowledge, or user preferences [121]. Failing to support such composite intents can lead to repetitive and less natural interactions affecting the user experience [109].

Traditional business process and service composition modeling and orchestration techniques are limited to support such conversations because they usually assume a priory expectations of what information and applications will be accessed and how users will explore these sources and services. Limiting conversations to a process model means that we can only support a small fraction of possible conversations [142]. While existing advances in Natural Language Processing (NLP) and Machine Learning (ML) techniques automate various tasks such as intent and slot recognition [50], the synthesis of API calls to support broad range of potentially complex user intents is still largely a manual, ad-hoc, and costly process [247]. Our goal is to bridge this gap by dynamically and incrementally synthesizing executable conversation model from NL conversations.

Informed by prior research and literature on conversational systems [50], in this chapter, we identify and characterize different types of conversation patterns to translate complex user utterances into operations that create composite (nested) states in a conversational state machine model [248]. These patterns mimics how a developer would have constructed workflows, leveraging conversation knowledge (i.e., slot values and

API element vectors), to realise some complex and decomposable user intents. More specifically, contributions in this chapter are summarized as follows:

- We identified a set of reusable composite dialogue patterns that naturally emerge when conversing with services.
- We extend an existing conversational model [248] to characterize state machine transformation patterns to support complex user intents.
- We provide validation and evaluation of the composite dialogue patterns presented in this chapter.

3.2 Related work

A number of techniques have been proposed to build task-oriented chatbots, including rule-based [23] and probabilistic models [100]. Main platforms such as Chatfuel [2] provide flow-based solutions to develop chatbots with zero coding using user interface elements. Research in this context includes the work by Lopez et al. [142], who propose a system that takes a business process model and generates a list of dialogue management rules to deploy the chatbot. Other platforms such as DialogFlow [5], on the other hand, provide ML based solutions. In addition to these solutions, a variety of ML models have emerged in research following two common architectures: *pipeline* and *end-to-end*. A *pipeline-based* model is built with a set of components, each responsible for a specific task such as tracking of intent/slot during conversations [51, 174]. *End-to-end* models, including end-to-end memory networks [253] and sequence-to-sequence models [150], read directly from a user utterance and produces a system action.

We identified a set of main limitations in the works above: First, rules-based approaches lack flexibility and require considerable development effort. Second, the use of existing probabilistic approaches and ML models such as memory networks becomes prohibitive due to the need for collecting huge and high quality training data. Third, flow-based approaches require the explicit definition of workflow, which is clearly unrealistic in large

scale and evolving environments. Furthermore, while ML approaches and platforms provide sophisticated support in term of intent/entities recognition and state tracking, they still far to handle conversations as either structured or unstructured processes. This is because they do not yet automatically support complex and decomposable user intents, where handling of intent requires information that is resulted from other intents either already processed or need to be. In addition, handling conversations as processes requires an advanced understanding of conversation context towards natural and straightforward dialogue experiences.

Similar to our approach, some advanced techniques like Devy [36], Iris [71], and Lu et al. [51] focus on more understanding of context especially by tracking required slots values from conversations history. However, since these slots values are derived only from conversation utterances they do not consider the knowledge of the heterogeneous APIs being used to converse with a wide variety of software-enabled services. This aspect is crucial to perform slot values inference accurately. In addition, these works do not propose any pattern that automates the identification of composite conversation flows. In summary, these efforts do not focus on augmenting conversations with knowledge that is essential for the superimposition natural language interactions over large number of evolving APIs. We therefore propose to consider API knowledge in the context and extend a conversational model to support a set of composite dialogue patterns that naturally emerge when conversing with services.

3.3 Human-Chatbot conversations

Conversations between a user and a chatbot are formulated as a sequence of utterances and responses. The conversation proceeds as a back-and-forth exchange. For example, a user might say to a chatbot, *"Send a message to @Sophia, tell her that I've just left so I'll be there in 10 minutes"*. The chatbot would respond by *"Your message to Sophia has been sent"*. As we discussed in Chapter 2, in order to answer user utterances, chatbots require to understand user intent (e.g., `send-message`) and extract slot-value pairs expressed by user (e.g., `(contact, Sophia)` and `(message, I've just left`

`so I'll be there in 10 minutes`) [49].

Studies on human conversation patterns [107,146,176] have shown that human-chatbot conversations can be divided into three types as shown in Figure 3.1.

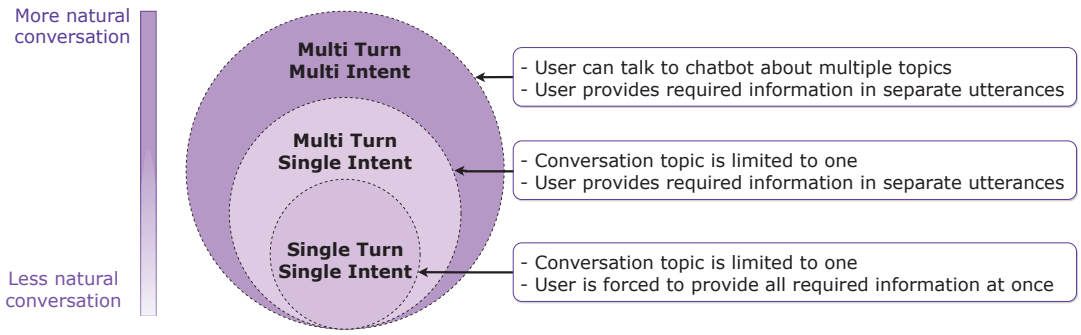

Figure 3.1: Types of human-chatbot conversations - from less to more natural [248].

Single Turn - Single Intent. The interactions between the user and the chatbot are straightforward. The user is expected to provide all required slot values in her/his utterance to fulfill her/his intent, thus the chatbot responds with a single, straightforward answer. For example, a user might ask a chatbot for a restaurant *"Show me some Italian restaurants near the Tower of Pisa"*, and the chatbot would simply respond with a list of options that match the user's requested criteria. Each user utterance is treated separately without using any knowledge from the conversation history. If any slot value is missing in the user utterance (e.g., `location`), the chatbot will not be able to fulfill the user intent. This type of conversation is *single intent* which means that the user and the chatbot converse only about one specific intent (e.g., `search-restaurant`) during the whole conversation.

Multi Turn - Single Intent. It is common for people to omit information in their daily conversations. This could happen for a variety of reasons, such as assuming the other person already knows the information, or simply forgetting to mention the information. For example, while chatting with a friend we may ask, *"Do you have any suggestion for a good Italian restaurant?"* without specifying the place where we are at (e.g., *"the Tower of Pisa"*). In order to answer our question, the friend needs to get more details by asking a question, *"Where are you?"*. Similar to this example, chatbot

needs to collect information that is scattered across multiple utterances to fulfill user intent. Thus, in this type of conversation, there are multiple turns to accomplish a specific user intent. *Dialogue management* component is responsible to maintain the conversation context and keep dialogue state to support multi-turn conversations.

Multi Turn - Multi Intent. Switching between intents or topics during conversations is a natural behavior for people. This can happen for a variety of reasons, such as a new idea or thought that comes up, or a desire to change the subject. Thus, in this type of conversation the user's intent continuously changes during the conversation. However, participating in a multi-intent conversation where information is scattered into multiple utterances, is a challenging task for chatbots. In the work [248], we proposed a conversational model to support *multi-intent and multi-turn* conversations. In the following sections, we first present this model (Section 3.4), then we explain how we extend this model to support composite user intents (Section 3.5).

3.4 State Machine Conversational Model

In order to support multi-turn multi-intent conversations, we proposed in the work [248] to represent User-Chatbot-Services conversations using an extended *Hierarchical State Machines (HSM)* model [238]. HSM is a well-known model that have been widely used to describe reactive behaviors of complex systems in a wide variety of areas [238]. This model is defined as a state machine where states can be ordinary states or composite states which are state machines themselves.

The proposed *state machine based conversation model* contains a set of states called *intent-states* representing user intents (e.g., `find-restaurant`), their slots (e.g., `city, food`) and actions such as API invocations (e.g., call `Yelp-SearchBusinesses` API method) to realize them. Each intent-state characterizes the fulfillment of specific user intent. Inside an intent-state, there can be *nested states* that represent situations that a chatbot may occupy in a given conversation (e.g., a chatbot-to-user question to resolve the value of a missing intent slot). Transitions between states are triggered when actions

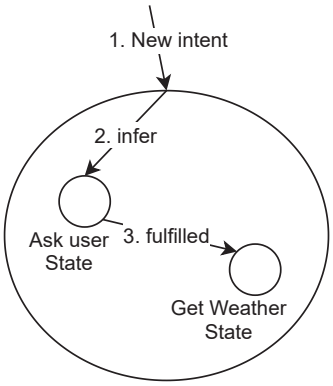

Figure 3.2: *Ask user* composite intent-state to fulfill `get-weather` intent.

are performed (e.g., a chatbot asks a question to a user to resolve a missing slot value) or upon detecting a new intent (i.e., switch from one intent-state to another). In the following, we describe the different types of states and transitions in this model:

Basic Intent State: We consider a state as a *basic* intent-state, when a user utterance carries all the required slots' values to fulfill the user intent. In other words, the chatbot has everything needed to perform the required action (e.g., API call). For example, given a user utterance (e.g., *"what is the weather in Paris?"*) with an intent (e.g., `get-weather`), chatbot invokes an API (e.g. `OpenWeatherMap` API) and returns a response to a user (e.g., *"we have light rain in Paris"*).

Nested Intent State: If a user utterance (e.g., *"what is the weather?"*) has a missing slot value, the chatbot needs to infer this missing value (e.g., value of the slot `location`) before it performs further actions to fulfill the intent. In this situation, the intent-state becomes *composite* intent-state and relies on other *nested* states to complete the intent. For example, as shown in Figure 3.2, the nested state `Ask user` is used by the chatbot to ask the user for the missing value of the slot `location` related to the intent `get-weather`.

New intent transition: This transition type refers to the movement between intent-states. The state machine transits to a new intent-state if the new user utterance corresponds to a new intent (i.e., detecting intent switch in conversations). For example, assuming that the state machine is in `send-msg` intent-state, then the user asks for

cinema. The user utterance *"I am also looking for cinema"* triggers a transition to move from the current intent-state `send-msg` to the new intent-state `find-cinema`.

Nested transition: This transition represents the movement of a state machine to a nested state. The state machine moves to a nested state if there is a missing slot value. The chatbot infers the missing value either from the context or by asking the user for it (i.e., `Ask user` nested state). For example, in Figure 3.2, the transition `infer` is a nested transition.

The motivation behind adopting the *state machine conversational model* is to reduce the complexity that may be caused by the number of states that are needed to specify interactions between users, chatbots, and services. Leveraging HSMs helps to factor out the common behaviors to reuse them across many states [238]. Thus, HSMs provide a very efficient way of sharing behavior so that they reduce the number of states needed for specifying User-Chatbot-Services interactions. In the next section we explain how we extend this conversational model to support a set of composite intents.

3.5 Composite Dialogue Patterns

In computer science, a pattern is defined as a reusable solution to a common problem within a specific context [97, 200, 213]. For example, design patterns are templates that can be applied to software design to solve recurring issues [97, 200, 213]. Every pattern has three main elements [200], which are: a pattern name, a context (or also called a problem), and a solution (or also called a description).

Pattern name. It is a descriptive term or label used to describe a specific problem and its solution. The pattern name lets us design at a higher level of abstraction.

Context. The context (or also known as a *problem*) describes a recurring set of situations in which the pattern can be applied. It explains the problem and its context. Generally, the context describes when to apply the pattern.

Description. It provides an abstract description of the pattern that can be applied

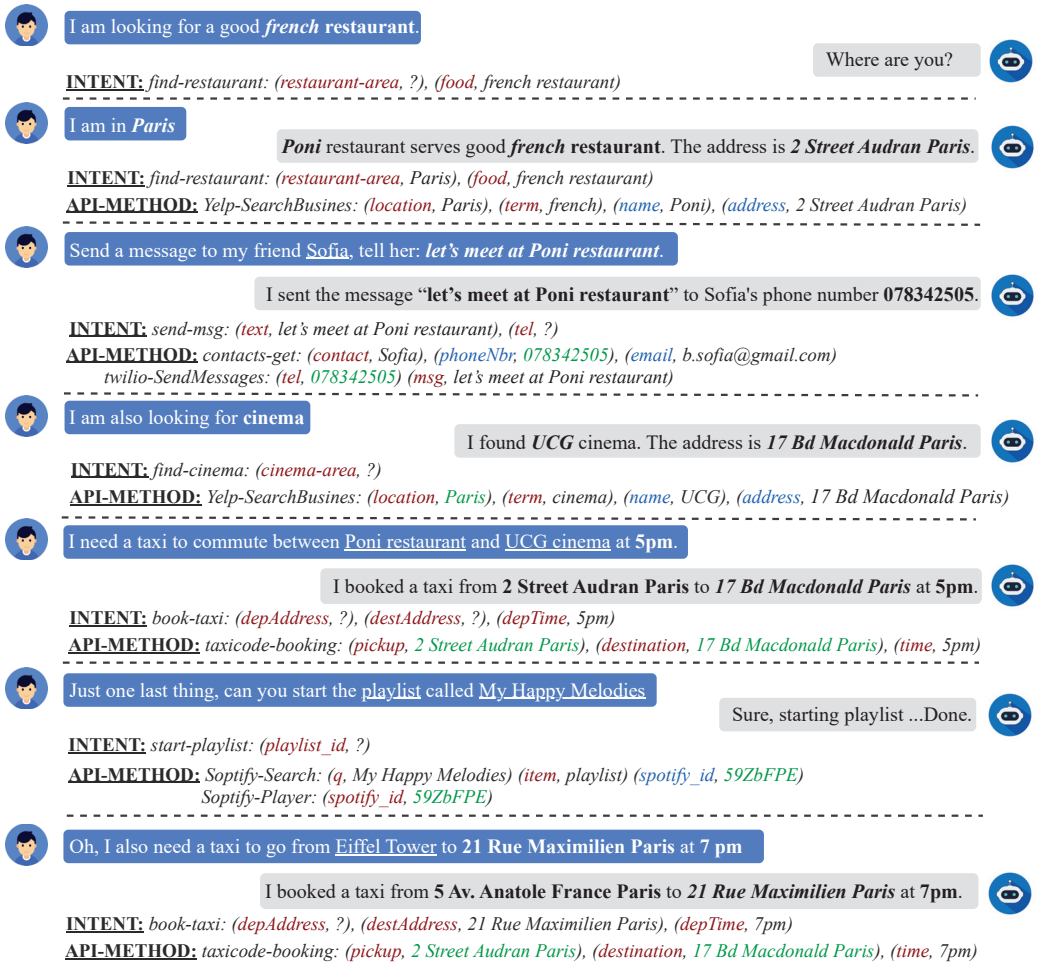

Figure 3.3: Example of multi-turn multi-intent conversation. After each turn, we illustrate the intent, its slot-value pairs, and the API call(s). The red slots/parameters are required input slots/parameters, the blue parameters are output parameters, and the green values are inferred values from different sources.

to resolve the problem. The solution does not describe a particular concrete design, instead, the pattern provides an abstract description of a design problem.

Inspired by existing workflow management systems and linguistic theory, in this section, we identify and characterize a set of reusable composite dialogue patterns that naturally emerge when conversing with services. These patterns mimics how a developer would have constructed workflows to realize some complex and decomposable user intents. In what follows, for each composite dialogue pattern, (i) we explain the problem that led us to propose the pattern, (ii) we give a description of the pattern, and (iii) we explain

through an example how we characterize the pattern in the conversational state machine model presented in Section 3.4.

3.5.1 Slot-value-flow pattern

Context. People are not always precise during their conversations. Sometimes they assume that their interlocutor already knows certain information from their previous conversations. Retrieving information from previous conversations can be obvious to human beings, however, it is difficult for chatbots to handle it. Incorrect inference of conversation flows arises from uncertainty about slot values and relationship between API elements across heterogeneous APIs (e.g., one API method uses `city` as a parameter while another use `location` as a parameter) and complex conversations. There are several parameters among multiple heterogeneous APIs methods that can share all or some of their values during the conversation [227]. Inspired by process models and workflow management where some of process activities' inputs are outputs from previous ones [69], we propose to enhance the chatbot to infer missing slot values from outputs of previous API calls.

Description. The composite pattern *slot-value-flow* allows the chatbot to resolve a missing value of an intent slot by extracting it from values of other parameter calls (e.g., an output parameter value of an already called API method). In other words, this pattern allows resolving a missing value of an intent slot by extracting it from a conversation history.

Example. Figure 3.4 illustrates an example of how the *slot-value-flow* pattern is represented in the conversational state machine model. Considering the user utterance *"I need a taxi to commute between Poni restaurant and UCG cinema at 5pm"* in Figure 3.3, the chatbot detects two missing slots' values (i.e., `depAddress` and `destAddress`) in the intent `book-taxi`. The chatbot leverages a *context knowledge service* to infer these missing values from the conversation history (e.g., it infers the value of `Taxi-depAddress` from `Restaurant-address` value). The chatbot creates a *slot-value-flow composite intent-state* in the conversational model as shown in Figure

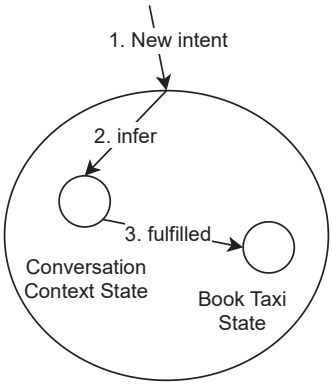

Figure 3.4: *Slot-value-flow* composite intent-state to fulfill `book-taxi` intent.

3.4. This composite intent-state relies on the `Conversation Context` nested state to infer the missing values and fulfill the intent `book-taxi`.

3.5.2 Nested-method pattern

Context. Sometimes, chatbots are repetitive and ask the user for missing information that can be retrieved by calling an API method. Similar to what is done in functional languages, some parameters are functions by themselves, which means that to get the value of a parameter, we need to call another function. Thus, instead of asking the user to provide missing information, the chatbot needs to check if it can retrieve it by calling an API method.

Description. The composite pattern *nested-method* allows the chatbot to resolve a missing value of an intent slot by triggering an API method to reuse its output values.

Example. Figure 3.5 illustrates how the *nested-method* pattern could be represented in the conversational state machine model. Considering the user utterance *"Send a message to my friend Sofia, tell her: let's meet at Poni restaurant"* in Figure 3.3, the chatbot detects that the value of `tel` slot is missing in the intent `send-msg`. The chatbot does not have to ask the user for the phone number to send the message because it can infer it by calling the API method `contacts-get`. Thus, the chatbot creates a *nested-method composite intent-state* in the conversational model as shown in Figure

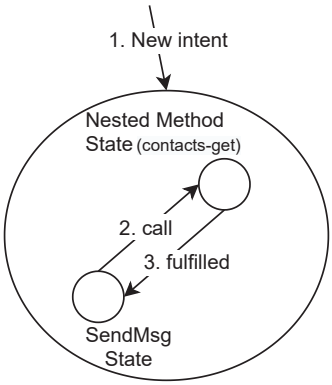

Figure 3.5: *Nested-method* composite intent-state to fulfill `send-msg` intent.

3.5. This composite intent-state relies on a nested state, called `Nested Method State`, to infer the value of the phone number and fulfill `send-msg` intent.

3.5.3 API-calls ordering pattern

Context. In REST API design, some methods require an API generated string, called *"id"*, as an input parameter to trigger methods. This *id* is an output of another method in the same API. To support these methods, bot developers have to implement an intermediately method that combines the sequence of API calls. Implementing new methods may resolve the problem, but in some cases, bot developers are constrained to implement multiple new methods that can combine more than two methods which is time consuming for them.

Description. The composite pattern *API-calls ordering* allows the chatbot to automatically map a user intent to a sequence of API calls to satisfy order constraints between methods of the same API.

Example. Figure 3.6 illustrates an example of how the *API-calls ordering* pattern is represented in the conversational state machine model. Considering the user utterance *"can you start the playlist called My Happy Melodies"* in Figure 3.3, the user wants to start a playlist, but the API method `Spotify-Player` requires a `spotify_id` as input to start the playlist which is missing. On the other hand, `Spotify-Search` is another

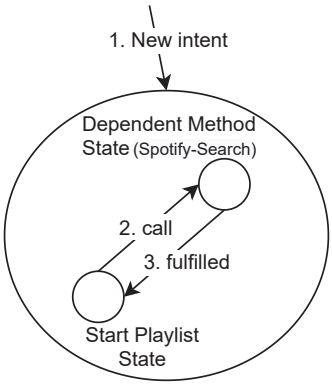

Figure 3.6: *API-calls ordering* composite intent-state to fulfill `start-playlist` intent.

Spotify API method that takes as input an item type (e.g., playlist, albums) and a keyword (e.g., *"My Happy Melodies"*) and returns an item Spotify Catalog information (e.g., owner, Spotify id, etc.). Thus, the chatbot first needs to call `Spotify-Search` to get the `spotify_id` then use this id to call `Spotify-Player` method. When the chatbot detects an API-calls ordering pattern, it creates an *API-calls ordering composite intent-state* in the conversational model as shown in Figure 3.6. This composite intent-state relies on a nested state, called `Dependent Method State`, to infer the value of the id.

3.5.4 Entity-enrichment pattern

Context. Humans are not always precise; they might refer to an entity mention that is common knowledge to inform a slot value. Thus, we believe that chatbots must have access to external data services to understand this common knowledge.

Description. The composite pattern *entity-enrichment* allows the chatbot to resolve a missing value of an intent slot from an external data service.

Example. Figure 3.7 illustrates an example of how the *entity-enrichment* pattern is represented in the conversational state machine model. Considering the user utterance *"I also need a taxi to go from Eiffel Tower to 21 Rue Maximilien Paris at 7 pm"* in Figure 3.3, the chatbot detects that the value of `depAddress` slot is missing in the intent `book-taxi`. The chatbot does not have to ask the user for the precise departure

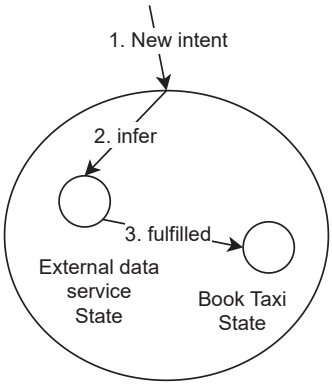

Figure 3.7: *Entity-enrichment* composite intent-state to `book-taxi` intent.

address because it can enrich the *"Eiffel Tower"* entity with additional information, such as its address, from an external data service (e.g., `Google-PlaceSearch`).

3.6 Validation

In this section we describe a study aiming at understanding the need, benefits, and effectiveness of supporting the proposed patterns. We investigate whether the proposed patterns naturally occur when conversing with services and perform a comparative analysis with alternative approaches focusing on the user experience.

3.6.1 Methods

Participants. Participants were recruited from our extended network of contacts, including students, colleagues and research groups. Invitations were sent via email, asking for volunteers to participate of the experiment. A total of 12 participants accepted to participate and completed the experiment as requested. This included master students, PhD students and senior researchers.

Experimental design. We followed a within-subjects design[1] to evaluate the proposed dialogue patterns and supporting services. Participants were tasked with interacting

[1]Study materials and in-depth results available at `https://tinyurl.com/25ad8jv6`

with two different chatbots, which were developed to capture the following experimental conditions:

- *DF-Baseline* : The baseline implements the standard conversational management support of traditional chatbot development platforms. It is developed using the underlying techniques of DialogFlow [5], including the DialogFlow NLU model, conversational model, and the Input-Output context mechanism.
- *SM-Patterns* : It supports the new proposed composite dialogue patterns and relies on the State Machine conversational model.

Besides the differences highlighted above, the two chatbots were built on the same foundation. They supported 15 intents collected from the DSTC8 dataset [174]. The DSTC8 dataset was chosen because it includes intents spanning multiple domains (e.g., Flights booking, Taxi booking, Finding places) which actually represents a challenging task for today's chatbots. In addition, these intents represent the most common tasks that users request in daily life and involve sequencing, nesting, API constraint, entity enrichment patterns as well. For the two chatbots, we use DialogFlow NLU service as NLU model because it is one of the most complete NLU models [44] to train chatbots.

We devised four main tasks, each comprising representative scenarios that catered to the proposed dialogue patterns:

T1 *Slot-value-flow pattern*: In this task[2], we requested participants to pick at least one of the following scenarios (a) planning a trip abroad, (b) planning a day program, or (c) paying back to a friend. These scenarios required orchestrating different services that would benefit from leveraging the ongoing context of the conversation (e.g., same locations or date).

T2 *Nested-method pattern*: Here, we asked participants to schedule a doctor's appointment on the first available spot. The dependency between the involved

[2]In the first task participants were asked to pick one of three available scenarios. Here we focus on *planning the day program*, as it was performed by all participants.

services (e.g., need to identify the doctor and check his/her availability before booking) favored the use of a nested-method pattern.

T3 *API-calls ordering pattern*: This task invited users to look for a restaurant with good ratings, thus requiring them to interact with two services (search for restaurants, and obtain reviews) linking the output of one to the input of the second. This scenario was designed to highlight the benefit of the automatically identifying these associations to serve more natural dialogue pattern.

T4 *Entity-enrichment pattern*: In this task, we asked participants to book a taxi *to* or *from* a known place (e.g., eiffel Tower) without giving the precise address. This scenario was designed to highlight the benefit of automatically inferring values from a external data services.

It is important to note that each scenario suggested the need for relevant services without imposing any specific conversation style or order.

Procedure. The study was conducted online with the support of an online form aggregating all the instructions. Before starting, participants provided their consent to participate and for their interactions with the chatbots to be recorded. After providing background information, participants then proceeded to perform the tasks with the two chatbots in a randomised order to avoid positional bias. For each task, participants were asked to describe the pros and cons of their experience with each chatbot, and to specify which one provided the better experience and why. The duration of the experiment was between 45-90 minutes.

Data processing and analysis. We performed a qualitative analysis of the experience with each chatbot. We performed a thematic analysis [39] of the open-ended participant feedback so as to identify emerging themes in their experience with the chatbots, and better characterise the reasons behind their preferred design. The conversation logs were also analysed to understand if participants naturally engage in conversations that leverage the proposed dialog patterns.

3.6.2 Results

T1. Slot-value-flow pattern. The large majority of participants (9/12) reported having a superior experience when interacting with the *SM-Patterns* chatbot as compared to the *DF-Baseline* . The qualitative analysis of participant feedback revealed two main reasons behind this preference. The dominant theme was the **efficiency of interactions** (9 participants), with participants expressing the *SM-Patterns* chatbot being *"quicker in getting an answer"* (P12) and being able to correctly infer missing values (e.g., *"I liked that it correctly understood my destination and I didn't have to input the address [from a previous turn]"*, P10). Another salient theme was the ability to enable more **natural conversations** (6 participants), with participants explicitly stating the *"experience of the conversation [being] more natural and human-like"* (P14). Participants also suggested improvements, notably in terms of being transparent (2 participants) about what information the chatbot was inferring from the context (e.g., *"render it clear what assumptions it is making, and allowing the user to accept / modify the values"*, P1).

The analysis of the conversation logs showed that the majority of participants (9 participants) engaged in conversations styles that took full advantage of this pattern, successfully referencing the context at least twice. Interestingly, the participants who showed preference towards the other chatbot engaged in conversation styles that to a lesser degree benefited of the slot-value-flow pattern, and instead formulated utterances that provided actual slot values in the requests (e.g., U: *"I want a taxi to [address]"*) instead of leveraging the context.

T2. Nested-intent pattern As in the previous task, the majority of participants expressed their preference for *SM-Patterns* (9/12 participants). The qualitative analysis of the feedback identified four main themes behind this preference. Participants referred to the chatbot's ability to **keep track of the user goal** (6 participants), stating that when engaging in a nested intent *"[the chatbot] remembered that I wanted to book appointment with a dentist (user goal)"* (P4) while the baseline would *"forget totally [what] I wanted"* (P3). Providing a **natural flow** was another emerging quality

attribute (4 participants), with participants expressing that the experienced *"flow felt natural"* (P6) while the baseline would force them to plan ahead (e.g., *"[DF-Baseline] would force me to think ahead about what services to call and in what order, as if defining a plan, instead of just interacting naturally with the chatbot and reacting to the information that is requested as the dialog progresses"*, P1). The chatbot was also perceived as **efficient** (5 participants), requiring *"less input for a correct answer"* (P14), while for a few it simply came down to being **effective** (2 participants), i.e., able to complete their task with the conversation styles they engaged in *"[SM-Patterns] was able to answer to my questions"* (P8).

An analysis of the conversation logs revealed that most participants (7/12) had naturally described a nesting-intent pattern in their interactions. Looking into the conversation logs of those who expressed preference for the baseline (3 participants) provided further insights. Interestingly 2 of these participants had not actually engaged in a nested-intent pattern, while the one who did had experienced problems in the formulation of the nested intent (i.e., the framing of the nested intent was not recognised by the NLU). Understandably, the preference of these participants was shaped by being unable to benefit from the proposed pattern. This highlights the need for integrating conversation repair strategies into this pattern.

T3. API-calls ordering pattern. All participants (12/12) reported having a better experience with the *SM-Patterns* chatbot describing it as being *"easier to follow up on related services"* (P1). Not surprisingly, the majority of participants (8 participants) commented on the ability to **hide technical details** as one of the main reasons for their preference, one participant citing that in the proposed scenario *"it successfully understood that I wanted a review from the selected restaurant without asking for the business ID"* (P7), whereas the technical details of the service as exposed by the baseline chatbot made it *"difficult to understand for someone who doesn't know what that means"* (P3). Providing a **smooth conversation flow** was another theme that emerged from the feedback on *SM-Patterns* , with participants mentioning that in comparison, interacting with the baseline chatbot felt like being *"caught in a loop"* (P8). Some participants summarised the positive experience by simply stating that the

chatbot was **effective**, working correctly or as expected (*"it gave the reviews correctly"*, P10).

The analysis of conversation logs showed that all but one participant (who deviated from the proposed scenario) described interactions that benefited from the API-calls ordering pattern. What this tells us is this pattern greatly aligns with the conversation styles and expectations of users.

T4. Entity-enrichment pattern. The majority of participants expressed their preference for *SM-Patterns* (10/12 participants). The qualitative analysis of participant feedback revealed main reasons behind this preference. Participants appreciated the chatbot's ability of inferring information from external services *"[it] found the address when I said Eiffel Tower"* (P6). In contrast, participants reported having to copy & paste previous values or google some information during their interactions with the baseline chatbot.

The analysis of conversation logs showed that all but two participant described interactions that benefited from the entity-enrichment pattern. These two participants spelled the known place (e.g., *"eiffel tower"*) incorrectly (e.g., *"book a taxi to the eifeltower"*), which led to the failure of the pattern detection for inferring values from an external data service. Therefore, handling mistakes when performing inferences is a situation that needs to be addressed. A pattern that detects errors in user utterances and fix them before doing the API call could support in this regard.

3.7 Conclusion

In this chapter we extended a conversational model to represent and reason about composite user intents. We also identified and characterized different types of dialogue patterns that endow bot platforms with reusable functionality to recognise compositional conversational flows and reduce the development complexity.

In the next chapter, we will delve into techniques that enable automatic recognition of

composite intents. These techniques will build on the concepts covered in this chapter, enabling chatbots to better understand and respond to complex user requests.

Chapter 4

Recognition of Composite Intents

Context Knowledge-aware Recognition of Composite Intents in Task-oriented Human-Bot Conversations

Contents

The content of this chapter is an extension of the work presented in [32]. In this work, we propose a new approach to recognize and realize composite user intents. The

proposed approach relies on a new rule-based technique that leverages both (i) natural language features extracted using existing NLP and ML techniques and (ii) contextual knowledge to capture the different classes of composite intents.

The rest of this chapter is organized as follows: We start with an introduction in Section 4.1. In Section 4.2 we discuss related work. In Section 4.3 we detail the *context knowledge service*, a service that we propose to provide the dialogue management with the required knowledge to support composite intent recognition. In Section 4.4, we present a hybrid approach to recognize and realize composite intents. Section 4.5 presents validation of the proposed approach, and finally we provide a conclusion in Section 4.6.

4.1 Introduction

Existing Natural Language Processing (NLP) and Machine Learning (ML) techniques have produced promising and useful results to recognize basic intents [229]. ML based techniques rely on the availability of massive amounts of annotated data. Using these techniques to recognize composite intents requires laborious, costly and hard to acquire training datasets. In addition, each time a new composite intent is identified, extending or producing a new dataset is needed as well. Therefore, more advanced and flexible techniques that cater for composite intent recognition are needed.

In the previous chapter, we focused on identifying and characterizing a set of composite dialogue patterns that naturally emerge when conversing with services. In this chapter, we focus on the *recognition* of complex intents in human-bot conversations.

We take the view that complex intent recognition could be significantly improved by considering composite dialogue patterns in addition to basic intent features. We propose an approach that relies on (i) existing NLP and ML techniques to extract natural language features (e.g., basic intents, slots' values) and (ii) a rule-based approach that leverages these features together with contextual knowledge, enabled by composite dialogue patterns and other metadata, to define *composite intent recognition rules*.

These rules capture in a *generic way* different classes of composite intents that may be expressed in user utterances. They enable a higher-level of abstraction that offers flexibility for an *extensible* library of composite dialogue patterns. When a new composite intent class is identified, a new rule template is added to recognize intents of this class from utterances. This approach requires to capture fairly complex context knowledge in addition to basic intents in order recognize complex intents. Thus, there is a need for advanced context representation and exploitation techniques that go beyond conversation history to include information inference that leverage metadata such as intent and API schemas (e.g, intents, slots, API methods) and relationships between their elements. Our contributions in this work are summarized as:

- We propose a hybrid approach that combines (i) natural language features, (ii) composite dialogue patterns, (iii) and contextual knowledge to capture different classes of composite intents in a generic way.
- We propose a Context Knowledge Service (CKS) to provide the contextual knowledge that is needed for defining the rules to recognize composite intents. This CKS consists of (i) a context knowledge model represented as a knowledge graph and (ii) a set of services facilitating the leverage of this knowledge.
- Empirical evidence showing the effectiveness and user experience of the CKS and composite intent recognition approach. The user study showed that endowing chatbots with the composite intent recognition allow less redundant and more natural interactions, as perceived by users and confirmed by performance metrics.

4.2 Related work

The work presented in this chapter is related to the state tracking process that aims to infer the dialogue state in terms of the user intent and its slot-value pairs during conversations [173]. Depending on the leveraged knowledge sources, existing state tracking approaches can be organized into history-based, schema-based, and linguistic patterns (LPs) based.

History-based approaches rely on the whole or window-size of the dialogue history to predict the dialogue state. Deep learning models including HRNN (Hierarchical Recurrent Neural Networks) [85], LSTM (Long-Short-Term-Memory) [79] and BERT (idirectional Encoder Representations from Transformers) [250] are utilised to encode the dialogue history. Other works [48, 179] leverage only on the previous dialogue states to predict the current state instead of taking the whole history. More advanced approaches [105, 163, 241] focused first on the learning of slot dependencies from the history and then incorporated them into the state tracking model, allowing it to infer slot values from similar slots. Most of the state tracking approaches either focused on recognizing only basic intents or ignored their recognition at all. Similar to some of these, we build upon advances in ML techniques to enable the recognition of basic intents but contribute a new rule-based approach to recognize composite intents.

Schemas-based approaches leverage schemas capturing the structural representation of conversation data to predict the dialogue state. Works like [55], [136], and [229] use a slot-level schema graph that captures dependencies between slots. The aim is to allow the state tracking model to infer slot values from similar slots. Other efforts like [135] leverage the backend database schema in state tracking model to allow slot inference from the database entities. These methods, however, work with chatbots integrated only with databases, where the inferred slot-value pairs are used to frame the query. Since our context knowledge model integrates API/Service schema, where each intent is associated with its corresponding API method, our approach can handle flows supported by software-enabled services.

The work [21] represents the dialogue state as a dataflow graph and the complex user intent as a dataflow program. For each user utterance, a trained model allows predicting the corresponding dataflow program. This approach relies on datasets where each utterance must be annotated with the corresponding dataflow program; however, it is not intuitive task to annotate utterances with programs.

The closest work to ours is [173] which introduced a unified schema defining an API as a combination of intents and slots. A BERT-based state tracking model then takes

this schema as input to enable the recognition of intents and inferring their slot-value pairs. The captured knowledge (i.e., the unified schema), however, can only help in the recognition of basic intents. In our work, we devise a context knowledge service mainly to capture the contextual knowledge that is required to recognize complex intents. In addition to the conversation history and intent/slot schemas, this knowledge includes API/Service schemas and enriched entities. The proposed approach exploits both this knowledge and basic intent features to recognize the complex intents.

LPs-based approaches leverages linguistic patterns that are drawn mainly from human conversations to handle some complex intents and inferring their slots [71,175]. For example, IRIS [71] draws on two existing LPs: *dependent questions* (i.e., one question depends on the answer to some subsequent questions), and *anaphora* (i.e., expressions that depend on previous expressions) to allow composition and sequencing of intents. These approaches, however, are not enough to capture complex intents that naturally emerge when conversing with services. For instance, while this utterance *"Send the message 'I will be at UGC cinema at 3 pm' to Sophia"* refers to a complex intent requiring a composition of two API methods (i.e., get-contact and send-message), it cannot be recognized by IRIS. The reason is that IRIS can recognize composition based only on linguistic features (e.g., a composition is recognized when a user answers with a new intent to the chatbot request for a missing slot). In contrast to the LPs-based approaches, we focus on using composite dialogue patterns that cater to the inherent features in interactions between humans, chatbots, and services in addition to the linguistic ones. These patterns are used to enable the contextual knowledge required by our approach to recognize complex intents.

4.3 Context Knowledge Service

Context can be defined as any information that can be leveraged from previous turns or other knowledge [108]. Maintaining the context is necessary in chatbots as it allows to keep continuity in the dialogue and avoid repetition, making interactions more natural [108]. However, inferring information from the context is challenging due to

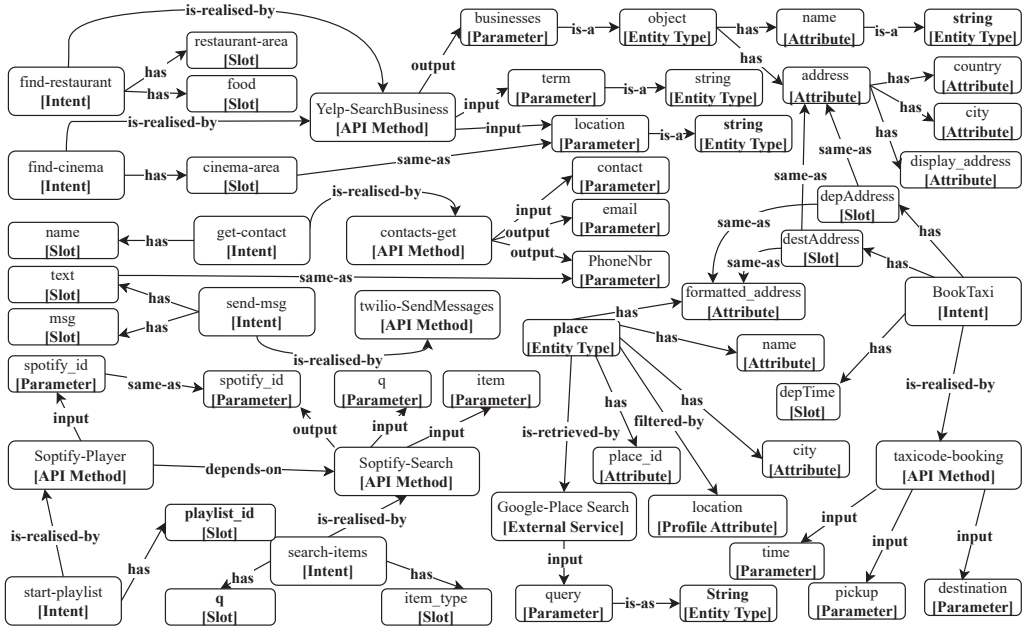

Figure 4.1: Context knowledge graph related to the conversation scenario in Figure 3.3. For clarity purpose we do not represent all nodes and edges.

multi-turn multi-intent conversations and heterogeneous APIs. To tackle this challenge, we proposed in [32, 33] a Context Knowledge Service (CKS) to provide the dialogue management with the required knowledge to support the recognition and realization of composite intents. The CKS has two main components: A context knowledge model (Section 4.3.1) and a set of services (Section 4.3.2) to leverage this knowledge.

4.3.1 Context Knowledge Model

The context knowledge model accomplishes the "magic behind the scenes" that enables chatbot to drive the underlying patterns from conversations and infer the set of information required in handling them. This context knowledge model consists of: (i) the metadata of chatbot schema, user profile and external services and (ii) the data stored as conversation progresses.

Metadata is represented as a context knowledge graph with a set of nodes and edges. Figure 4.1 shows an excerpt of a context knowledge graph related to the conversation in Figure 3.3. We distinguish between 8 node types: `Intents`, `Slots`, `Methods`,

`Parameters`, `Entity Types`, `Attributes`, `External Services`, and `Profile Attributes`. `Methods` refer to concrete API methods being invoked to fulfill user intents (e.g., Yelp-SearchBusiness). `Parameters` designate either input or output parameters of a method. `Entity Types` refer to the type of slots, parameters, and attributes (e.g., string, person). `External services` refer to external data services (e.g., Google-PlaceSearch) that can be used to retrieve certain attribute values. Often, a data service has an input parameter that takes a text query, which in our case will be filled by the entity mention (e.g., *"Eiffel Tower"*).

The key intuition behind including external data services is to endow the chatbot with the capability of enriching entities with additional information from these services. However, an external data service may return several entities for a given mention. Thus, having a mechanism that links the entity mention to its corresponding entity in the data service is necessary. This is where the user profile comes into play. User `Profile Attribute` such as *location* and *preferences* can be used as filters to select the appropriate entity.

Furthermore, there are 9 edges types: `is-realized-by`, `input`, `output`, `has`, `is-a`, `depends-on`, `same-as`, `is-retrieved-by`, and `filtered-by`. The edge `is-realized-by` denotes that an intent is realized by an API method. The edges `input` and `output` denote that a parameter is an input or an output of a method (e.g., *"location"* is an input parameter of *"Yelp-SearchBusiness"* method). The edge `input` can also be defined between an external service and its input parameter (e.g., *"query"* is an input parameter of *"Google-PlaceSearch"* service). The edge `has` denotes that an intent has a slot (e.g., *"find-restaurant"* intent has *"restaurant-area"* as slot), or an entity type has an attribute (e.g., an *"address"* entity type has the attribute *"city"*). The edge `is-a` is defined between a slot, an attribute or a parameter and an entity type (e.g., location parameter is a string). The edge `depends-on` denotes that an API method depends on another method to get the required `id` parameter value. The `same-as` edge is generated between slots, parameters, and attributes if they have the same type (e.g., *"restaurant-area"* slot is same-as *"location parameter"*). The edge `is-retrieved-by` is defined between an entity type and an external service (e.g., the entity type *"place"*

can be retrieved by *"Google-PlaceSearch"* service). The edge `filtered-by` is defined between an entity type and a profile attribute, meaning that entities of that entity type can be filtered by the given attribute (e.g., *"place"* entities can be filtered by *"location"* profile attribute).

The nodes and edges including `is-realized-by`, `input`, `output`, `has`, `is-a` are generated from the chatbot schema. We leverage on word embedding techniques to automatically generate `same-as` edges [181]. More precisely, we compute the corresponding vector embedding for each slot, parameter and attribute. Then, we measure the cosine similarity between each pair of slots, parameters, and attributes [181]. If the similarity of each pair exceeds a predefined threshold, a `same-as` edge is generated between them. The generation of `filtered-by` edges is based on computing the cosine similarity between the vector embedding of each pair of entity type attributes and profile attributes. If the similarity exceeds a predefined threshold, a `filtered-by` edge is added between the entity type and the similar profile attribute. For example, assume that a user profile is defined by this set of attributes (`gender`, `location`, `dietary`) and the entity type `place` has the attribute `city` among others, a `filter-by` edge will be added between `place` and only `location` since it is similar to `city`. We assume that the external data services, the user profile attributes, and the edges `is-retrieved-by` and `depends-on` are specified by the chatbot developer.

Data includes relevant information that should be memorized during user-chatbot conversations for later reuse. Two memory structures are used to store this data: *Local Context Memory (LCM)* and *External Context Memory (ECM)*. The LCM keeps track of all the traces related to each intent fulfillment. This includes the utterance, the intent, the method call, alongside with its timestamp, its inputs, and its outputs. The ECM, on the other hand, keeps track of all entities mentions in user utterances. It also provides all information that external data services extract to enrich these entities mentions. We structure ECM in terms of entities; each is associated with its mention (e.g. *"Eiffel Tower"*), entity type (e.g., `Place`), and its retrieved attribute values. The ECM is continuously updated as new entity mentions are detected or new attribute

values are retrieved.

4.3.2 CK services

The CKS features a set of services which are devoted to supporting the inference of slot values and providing the contextual knowledge that is needed to recognize and realize composite intents. In what follows, we define each of these services:

History search: This service allows inferring slot values from the conversation history. It is enabled by the endpoint "`CKS/history? ms & u`" which takes as inputs the missing slot `ms`, and the utterance `u` and returns the value of the slot `ms` when possible. First, the service extracts the relevant entity-mention pairs from the utterance. An entity-mention is relevant if the extracted entity has an attribute same-as the missing slot. Consider the user utterance *"I need a taxi to commute between Poni restaurant and UCG cinema"*, the entity-mention `(restaurant, Poni)` is relevant because the entity `restaurant` has an attribute `address` same-as the missing slot `taxi-depAddress`. Second, the service rewrites the utterance by replacing each mention with the corresponding attribute's value. For example, the previous utterance will be *"I need a taxi to commute between [Poni-address] and [UCG-address]"*. The rewriting is important to know if the value of `taxi-depAddress` is the restaurant or the cinema address. The service then extracts the missing value from the new utterance. If there is no relevant entity-mention in the utterance, the service returns the most recent value of the parameters same-as the missing slot. For example, in utterance *"I am also looking for cinema"*, the slot's value `cinema-area` is missing and there is no entity-mention, so the service returns the value of the restaurant's area.

Entity enrichment: This service allows enriching entity attributes from external data services. It is enabled by the endpoint "`CKS/invoke_external_service? s & em & a`". Consider the utterance *"I also need a taxi to go from Eiffel Tower"*, the service takes as inputs: the external data service `s`: `Google-PlaceSearch`, the entity-mention `em: (place, Eiffel Tower)`, and the attribute `a`: `address` and it returns the value

of the attribute `a`. To obtain the attribute value from the appropriate entity, three steps are followed. First, the service invokes the external data service related to the given entity-mention `em`, which returns a set of entities. Then, it filters the returned entities by discarding any entity, whose similarity with the entity-mention `em` is less than a predefined threshold and it does not contain the target attribute value. The similarity is computed on the basis of the cosine distance between the embedding vectors of the entity-mention `em` and the name of the entity returned by the external service. After this step, if only one entity is returned, the service retrieves the target attribute value from it. Otherwise, in order to identify the right entity, the service proceeds a second filtering step based on the filter attributes related to the mention entity type in the metadata. This filter step is expected to return one entity that matches the most of filters while giving a high priority to the location filter. In other words, if the location attribute is among the filter attributes, all entities that do not satisfy it will be discarded despite that they may satisfy other attributes.

Nested method identification: This service allows identifying an API method that needs to be invoked to obtain the missing slot value. It is enabled by the endpoint "`CKS/nested_method? ms & set_em`". Consider utterance *"Tell Sofia let's meet at Poni restaurant"*. The service takes as inputs the missing slot `ms`: `tel`, and the set of detected entity-mentions `set_em`: `{(person, Sofia)}`. It then gets from the metadata the methods that have an output parameter same-as the slot `ms`. For example, the service gets the set of methods `{contacts-get, businessDetails-get}` where the missing slot `tel` is the same-as one of the outputs of `contacts-get` (i.e., `phoneNbr`) and also the same-as one of the outputs of `businessDetails-get` (i.e., `phone`). The service relies on the detected entity-mentions to select the relevant method from the set of methods. For example, in contrast to the method `businessDetails-get`, the method `contacts-get` has an input parameter `contact` same-as to one of the detected entity `person`. Thus, the service selects `contacts-get` as the nested method and returns it along with its input values {(`contact`, `Sofia`)} and one of its outputs `phoneNbr` that is same-as the missing slot `tel`.

Dependent Method identification: This service allows identifying a dependent method to get a value of an id. It is enabled by the endpoint "`CKS/dependent _method?i`" where `i` is a given intent. This endpoint first gets the method `m1` that realize the intent `i`. Then, it gets the dependent method `m2` where `m1` depends on `m2`. For example, in utterance *"can you start the playlist called My Happy Melodies"*, the endpoint takes the intent `start-playlist` as input and returns: the dependent method `Spotify-Search`, its input parameters `{q, item}`, and its `id` output parameter `spotify_id`.

4.4 Composite Intent Recognition Rules

In this Section, we focus on the *recognition* and *realization* of composite intents in human-chatbot conversations. We take the view that composite intent recognition could be significantly improved by considering composite dialogue patterns in addition to basic intent features. In [32], we proposed an approach that relies on (i) existing NLP and ML techniques to extract natural language features (e.g., basic intents, slots' values) and (ii) a rule-based approach that leverages these features together with contextual knowledge, enabled by composite dialogue patterns and other metadata, to define composite intent recognition rules. These rules enable a higher-level of abstraction that offers flexibility for an *extensible* library of composite dialogue patterns. When a new composite intent class is identified, a new rule template is added to recognize intents of this class from utterances. In nutshell, these rules capture in a generic way different classes of composite intents that may be expressed in user utterances.

We express a rule using a combination of natural language features (provided by *dialogue act functions*) and contextual knowledge (provided by *context metadata functions*). In what follows, we first define functions that we use to specify the rules (Section 4.4.1), then we specify the rule of each dialogue pattern defined in Chapter 3 (Section 4.4.2).

4.4.1 Functions

Functions are the primitives that we use to define the rules. We consider function input and output types to be standard data types found in common programming languages. This includes simple data types such as `string` and `boolean`; as well complex data types such as `Tuple` or `Set`. Thus, we can leverage the standard operators designed for these data types. We distinguish two types of functions: *dialogue act functions* to capture natural language features and *context metadata functions* to capture contextual knowledge. These functions are offered by the NLU and the CKS, respectively.

Dialogue Act functions are important to identify hidden actions in user or chatbot messages. Whether the user is providing information, or asking a question, or the chatbot is asking for a missing information, or providing suggestions, are all hidden acts in user or chatbot messages. We focus on two dialogue act functions: `INTENT_OF()`, which identifies the intent `i` expressed in a given utterance `u`, and `SLOT_VALUE()`, which returns the value of a given slot `s` recognized in the utterance `u` or `NULL` if no value of `s` is recognized in the utterance `u`.

Context Metadata functions are important to capture the contextual knowledge. They allow to access and query the metadata graph defined in Section 4.3.1. For example: `GET_SAMEAS_PARA()` is a metadata function that returns a set of parameters that are the same-as a given slot `s`; `DEPENDS_ON()` returns a method name $\mathtt{m}_b$ given a method name $\mathtt{m}_a$ where $\mathtt{m}_a$ depends on $\mathtt{m}_b$, or it returns `NULL` if there is no dependent method; `GET_ATT()` returns the set of attributes of a given entity `e`.

4.4.2 Rules

A rule consists of *trigger* and *action* clauses. The trigger clause specifies the conditions that need to be verified to recognize composite intents. Then, the sequence of operations specified in the action clause are executed to realize the related composite intent. The following statement specify a rule:

Table 4.1: Examples of boolean functions to express triggers

Functions	Inputs	Description
`IS_NEW_INTENT()`	u: string	returns `true` if the identified intent in the utterance `u` is a new intent.
`HAS_MISSING_SLOT()`	u: string s: string	returns `true` if the value of the given slot `s` is not recognized in the utterance `u`.
`HAS_SAMEAS_PARA()`	s: string	returns `true` if there is at least one parameter that is the `same-as` the slot `s`.
`EXIST_NESTED()`	s: string	returns `true` if there is at least one output parameter that is the `same-as` the slot `s`.
`IS_DEPENDENT()`	i: string	returns `true` if the method that realize `i depends on` another method.
`HAS_SAMEAS_ATT()`	set_em: set s: string	returns `true` if at least one entity in `set_em` has an attribute that is the `same-as` the slot `s`.

`Rule` "name of the rule" **when** `trigger` **then** `action`

Triggers are expressed as boolean conditions over functions, including dialogue act and metadata functions. Table 4.1 provides examples of boolean functions that are used to define triggers. Conditions may be combined using conjunction operator (`AND`). The action is a sequence of operations. For instance, an operation can be an assignment of a value to a given variable, or an invocation of a CKS service. In what follows, we describe the rule of each pattern introduced in Chapter 3:

Slot-value-flow Rule. Figure 4.2 shows the specification of the rule related to *slot-value-flow* composite intent. The first condition checks if the identified intent `i` is a new intent. The second condition checks if the value of the slot `ms` is missing (i.e., not recognized in `u`). The third condition checks if there is at least one already fulfilled parameter that is the `same-as` the slot `ms`. If the three conditions are satisfied, the chatbot: **(1)** invokes the *history search* CKS service to get the missing value, **(2)** adds this value to the set of slot-value pairs of the intent `i`, and **(3)** invokes the method that realize the intent `i`.

Nested-method Rule. Figure 4.2 shows the specification of the rule related to *nested-method* composite intent. The first two conditions are the same as *slot-value-flow* rule. The third condition checks if there is at least one output parameter that is the `same-as` the slot `ms`. If the conditions are satisfied, the chatbot: **(1)** invokes the *nested method identification* CKS service to identify: the nested method $\mathtt{m}_{nes}$, its input values

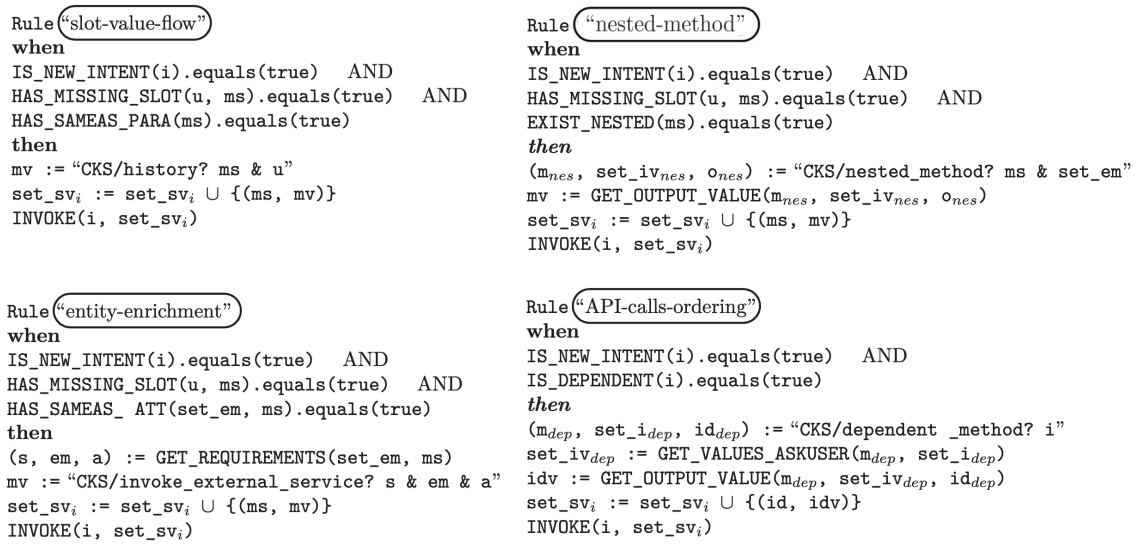

Figure 4.2: Rules of composite dialogue patterns introduced in Chapter 3.

$\texttt{set\_iv}_{nes}$, and its output parameter $\texttt{o}_{nes}$. Then, the chatbot **(2)** invokes the method $\texttt{m}_{nes}$ to get the value of $\texttt{o}_{nes}$, **(3)** uses this value as a value for the slot `ms`, and **(4)** invokes the method that realize the intent `i`.

API-calls ordering Rule. Figure 4.2 shows the specification of the rule related to *API-calls ordering* composite intent. The first condition is similar to the first condition of the *slot-value-flow* rule. The second condition checks if the method that realize the intent `i` depends on another method. If the conditions are satisfied, the chatbot: **(1)** invokes the *dependent method identification* CKS service to identify: the dependent method $\texttt{m}_{dep}$, its inputs $\texttt{set\_i}_{dep}$, and the id parameter $\texttt{id}_{dep}$. Then, the chatbot **(2)** calls `GET_VALUES_ASKUSER()` to get the input values of the method $\texttt{m}_{dep}$ by extracting them from the utterance, the history, or by asking the user. After getting the input values, the chatbot **(3)** invokes $\texttt{m}_{dep}$ to get the value of $\texttt{id}_{dep}$, **(4)** uses this value as a value for the parameter `id`, and **(5)** invokes the method that realize the intent `i`.

Entity-enrichment Rule. Figure 4.2 shows the specification of the rule related to *entity-enrichment* composite intent. The first two conditions are the same as *slot-value-flow* rule. Given a set of entities mentions `set_em`, extracted from `u`, the third condition checks if there is at least one attribute of an entity-mention that is the `same-as` the

slot `ms`. If the conditions are satisfied, the chatbot: **(1)** calls the metadata function `GET_REQUIREMENTS()` to get the following information: the related service `s`, the entity-mention `em`, and the attribute `a`. Note that this function chooses one entity-mention from `set_em` based on the one that has an attribute `same-as` the slot `ms`. Then, the chatbot **(2)** invokes the *entity enrichment* CKS service to get the value of the attribute, **(3)** uses this value as a value for the slot `ms`, and **(4)** invokes the method that realize the intent `i`.

4.5 Validation

The first objective of the study was to explore the *effectiveness* and limitations of (i) the proposed CKS (i.e., its capability of inferring slots' values correctly and reducing unnecessary interactions) and (ii) the composite intent recognition approach (i.e., its capability of recognizing correctly the composite dialogue patterns described in Chapter 3). The second objective was to evaluate the user experience (i.e., *naturalness*, *repetitiveness*, *understanding*) in interacting with a chatbot improved with CKS and composite intent rules.

4.5.1 Methods

4.5.1.1 Experimental design

Participants were recruited via email from our extended network of contacts. The call for volunteers resulted in a total of 20 participants. We prepared an evaluation scenario that required participants to interact with a set of API methods through a chatbot to plan an evening activity. Participants were asked to complete four different tasks in this scenario (T1: checking the weather and searching for restaurants, T2: booking a restaurant table, T3: booking a taxi, and T4: sending a confirmation message to the travel partner). The tasks were designed to leverage the type of support provided by the CKS, if the composite dialogue patterns were to be effectively recognized (T1:

inferring slot value from conversation history, T2: identifying dependent method, T3: using an external data source, and T4: identifying nested method). We followed a within-subjects design,[1] tasking participants to interact with two chatbots representing the following experimental conditions:

- *DM-Baseline*. The baseline implements a standard conversational management, without composite dialogue patterns and CKS support.
- *DM-CKS*. This chatbot is implemented with the composite intent rules and CKS support.

The two chatbots relied on the same NLU implementation (in DialogFlow [5]), bot interface, and differed only in the composite intent rules and CKS support.

4.5.1.2 Procedure

The study was conducted online. Participants received a link to an online form that included an informed consent, all the instructions, links to the chatbots and feedback required. In the study, participants were introduced to the evaluation scenario and tasks, and were asked to perform those tasks with the two chatbots. The order in which the chatbots were presented to users was counterbalanced to avoid positional bias. For each chatbot, participants were asked to provide open-ended feedback on the pros and cons of their experience.

The last part of the study then asked participants about their preferred chatbot, the reason why, and a quantitative feedback on their user experience. We adopted the user experience questions from the *Chatbot Usability Questionnaire (CUQ)* [103]., to get feedback on the perceived *naturalness* (i.e., ability of the chatbot to fulfill user tasks in human-like conversations), *repetitiveness* (i.e., ability of the chatbot to avoid redundant questions) and *understanding* (i.e., ability of the chatbot to interpret user requests). The duration of the study was between 15-20 minutes.

[1]Study materials and in-depth results available at `https://tinyurl.com/study-materials`

Table 4.2: Chatbot performance for each task according to relevant metrics. Values in bold denote best performance. Percentages denote the relative performance with respect to the reference (optimal) scenario.

Task (service)	DM-Baseline			DM-CKS			
	M1 (TURNS)	M2 (PROMPTS)	M3 (SLOTS)	M1 (TURNS)	M2 (PROMPTS)	M3 (SLOTS)	M4 (PATTERN)
T1 (history)	59.70%	21.18%	49.24%	**98.04%**	**90%**	**96.97%**	**95%**
T2 (dependent)	61.18%	42.11%	17.86%	**92.86%**	**95.24%**	**95.24%**	**94.44%**
T3 (external)	46.97%	32.39%	3.17%	**91.18%**	**88.37%**	**95.24%**	**95%**
T4 (nested)	52.97%	20%	39.41%	**96.55%**	**80%**	**96.55%**	**95%**
Mean	55.21%	28.92%	27.42%	**94.66%**	**88.4%**	**96%**	**94.86%**

4.5.1.3 Data analysis

We performed an analysis of conversation logs so as to assess the effectiveness of the CKS and the composite intent rules (CIR). These are calculated in relation to optimal conversation scenarios[2] that we designed based on participants conversations. The CKS effectiveness is calculated by considering the following metrics: number of (M1) conversation turns, (M2) prompts asking for missing slot values, and (M3) missing slot values correctly inferred. The effectiveness of the CIR (only available in DM-CKS) is calculated by considering the number of (M4) composite intents correctly detected. These metrics are calculated per user conversation, aggregated (mean) and then used to compute the relative performance against the optimal scenario. We also performed a qualitative analysis of open-ended responses and conversation logs to contextualise the results from the metrics and identify limitations.

4.5.2 Results

4.5.2.1 Effectiveness of CKS and CIR

Table 5.2 shows the relative performance by task of both chatbots DM-Baseline and DM-CKS in relation to the optimal reference scenario. For the four tasks, we can see that DM-CKS chatbot experienced a boost in performance for M1 and M2 metrics (mean across tasks 94.66% and 88.4% respectively), approaching the efficiency in terms of number of turns and prompts of the reference ideal scenario. This level of

[2]Scenarios assuming ideal accuracy of slot-value inference and intent recognition.

performance is possible due to the accuracy of the slot value inference (M3) performed by the CKS services supporting each task – a mean relative performance across tasks of 96%. In contrast, not having the support of the CKS services lead the DM-Baseline chatbot to perform poorly in comparison, with the best performance being at around 37.18% for the metrics considered. These results provide evidence for the benefits and effectiveness of the CKS support.

Table 5.2 also shows the relative performance of recognizing composite intent (M4) by the chatbot DM-CKS in relation to the reference scenario. By analyzing the conversations, we noticed that the recognition error of the composite dialogue patterns is mostly caused by the detection error of the correct intent by the NLU during the conversation. For example, if the NLU detects the intent `search-restaurant` instead of `book-taxi`, in the utterance *"I want to go to this restaurant"*, this will lead to an error in detecting the slot-value-flow pattern that takes the restaurant address as the destination address. However, for the four tasks, we can see that DM-CKS chatbot is close to the reference scenario with a mean relative performance across tasks of 94.86%.

4.5.2.2 User experience

All but one participant (19/20 participants) expressed a preference towards the DM-CKS chatbot as opposed to the baseline. The one exception was due to NLU limitations in recognizing user expressions that led to the enactment of the wrong services. The feedback to the specific user experience questions, as well as the open-ended feedback, highlighted the reasons behind the preference. Participants agreed with DM-CKS interactions describing *naturalness* (14/20), less *repetitiveness* (16/20) and *understanding* (15/20), whereas the baseline was poorly rated on these fronts (1/20).

Interestingly, these qualities were linked to the CKS support, such as the ability to infer missing slot values from conversation history (e.g., *"saying that the drop-off address was the restaurant I have just booked was enough"*, P1), or from external services (e.g., *"[it] found the address when I said Eiffel Tower"*, P6). The ability to handle composite intents also emerged as a defining feature (e.g., *"[DM-CKS] is capable of undertaking*

complex tasks and retaining previous information", P16). In contrast, participants reported having to copy & paste previous values or google some information during their interactions with the baseline chatbot.

4.5.2.3 Limitations

The conversation analysis revealed some limitations in supporting the natural language interaction described by the users:

Enumerating entities when the number of entities is expected. In the context of T2, when asked *"For how many people do you want to book a table?"*, some participants would respond with *"For me and my friend"*. The chatbot could not infer the number of seats from the participant's utterance because it expected to extract a number. This would be an acceptable answer in a natural conversation, and represent as a new type of inference that needs to be considered.

Introducing typos when providing slot values. When booking a taxi in the context of T3, some participants spelled *"Eiffel tower"* incorrectly (e.g., *"book a Taxi from the Eifeltower"*), which led to the failure of the CKS service for inferring values from an external data source. Handling mistakes when performing inferences is a situation that needs to be addressed.

Coreferences in user utterances not resolved. Another reason why the slot value inference failed at times are limitations inherent to the adopted coreference model. For example, after booking a restaurant, a participant asked the chatbot *"can you get me a taxi there?"*. The NeuralCoref model [9] failed to replace *"there"* by the restaurant present in the conversation context. Testing alternative coreference models, such as Stanford's CoreNLP [151], could improve the slot value inference mechanism.

4.6 Conclusion

In this chapter, we proposed reusable and extensible rule-based technique that uses a sophisticated context service to recognize and realize composite intents. We believe that our approach charts novel abstractions that unlock the seamless and scalable integration of natural language-based conversations with software-enabled services. We devised a novel composite intent recognition that allows the incremental acquisition of rule templates to identify composite intents from basic dialogue acts and context features. The contextual knowledge required at run-time to recognize composite intents and infer slot values from user-chatbot conversations is extracted from conversation history, enriched entities, intents and API schemas and represented in graph structure.

While we focused on interactions between user, chatbot and simple services (*Intent-SingleAPI*), other interactions with composite services (*Intent-CompositeAPI*) can be captured. In the next chapter, we propose a new concept to represent natural language conversations between the user, the chatbot, and composite services.

Chapter 5

Process-oriented Intents

Process-oriented Intents for Superimposition of Natural Language Conversations over Composite Services

Contents

The content of this chapter is an extension of the work presented in [31]. This work focuses on the superimposition of task-oriented assistants over *composite services*. We propose *Human-bot-Process interaction acts* that are relevant to represent natural language conversations between the user and multi-step processes. In doing so, we enable human users to perform tasks by naturally interacting with service orchestrations involving multiple actions.

The rest of this chapter is organized as follows: We start with an introduction in Section 5.1. In Section 5.2 we discuss related work. Section 5.3 provides preliminaries, a motivating scenario and requirements. In Section 5.4, we present the general architecture. Section 5.5 presents the characterization and recognition of *process-oriented intents*. Section 5.6 presents validation of the proposed approach, and finally we provide a conclusion in Section 5.7.

5.1 Introduction

Task-oriented conversational services (or simply chatbots) emerged as engines for transforming online service-enabled digital assistance and powering natural interactions between humans, services, and things [192]. Recently, organizations leveraged task-oriented chatbots in a variety of assistance tasks, including healthcare, education and e-commerce. For instance, the augmentation of process-enabled automation with AI-enabled conversational assistants emerged as a promising technology to make process automation even closer to users (e.g., customers, workers) [25, 35]. This evolution promises to increase the benefits of automation by simplifying access and reuse of concomitant capabilities across potentially large number of evolving and heterogeneous data sources, applications and things [18, 35]. Integrating AI-enabled task-oriented

conversational assistants and process-enabled automation involves advanced machine learning, entity recognition, and Natural Language Processing (NLP) techniques [25]. A key NLP task in this context is intent recognition, i.e., (i) understanding user utterances in natural language and recognizing user intent corresponding to tasks that the user wants to accomplish, (ii) extracting relevant intent input slot values from user utterances, and (iii) trigger commands that process user intents and perform conversations with users.

In the previous chapters, we proposed various techniques for the superimposition of task-oriented conversational services on top of Application Programming Interfaces (APIs) [32, 33, 248]. More precisely, we proposed conversation state machines to represent multi-turn and multi-intent conversations between users and API-enabled services [248]. We identified and characterized a set of composite dialogue patterns that naturally emerge when conversing with services [33]. We proposed a hybrid intent recognition approach that combines (i) basic intents recognition from natural language user utterances and (ii) contextual knowledge, including API schemas and API integration patterns (e.g., data flow between API methods, nested API methods, dependency constraints between API methods), to recognize different classes of composite intents like a sequence of API calls and nested API calls [32].

In this chapter, we focus on the superimposition of task-oriented assistants over *composite services*. In doing so, we enable human users to perform tasks by naturally interacting with service orchestrations involving multiple actions (e.g., sequence of API methods). Orchestrating human–machine conversations over composite services requires rich abstractions and knowledge to: (i) interact with a multi-step processes (e.g., accessible through a business process language engine or service middleware [69]) using natural language utterances, (ii) automatically recognise nuanced, context sensitive and possibly ambiguous process-aware user intents including starting a new task, inquiring about task progress, switching from one task to another and exceptional behavior such as canceling or updating tasks. Specifically, we identify fine-grained *Human-bot-Process (HP) interaction acts* that are relevant to represent natural language conversations between user and multi-step processes. In a nutshell, interaction acts are dialogue acts

that characterise process-oriented intents[1] in user utterances expressed in natural language. For instance, a *task creation interaction act* involves creating a new process instance that performs the requested task; a *task cancellation interaction act* involves the cancellation of a previously performed task. These interaction acts should capture both the expected (e.g., purchase a product) and exceptional behaviors (e.g., return product) of a process. Our contributions in this work are summarized as follows:

- We propose *Human-bot-Process interaction acts* as abstractions to characterize possible *process-oriented intents* in conversations between users and multi-step processes.
- We identify common process-oriented intents. We believe that interactions with process-aware chatbots require more fine-grained intents for accomplishing tasks, such as initiating, analysing, monitoring and controlling process instances through natural language utterances. We present these intents and describe their benefits through practical scenarios.
- We devise an approach that combines user utterance features (e.g., user intents, HP interaction acts) and process knowledge (e.g., process schema, process instance correlation attributes, information about possible exceptions) to recognize process-oriented intents from user utterances.
- We discuss both crowd-based and automatic paraphrasing strategies [168] to collect utterance paraphrases and build a dialogue dataset to train process-oriented intent recognition models.
- Empirical evidence showing the effectiveness of the proposed approach in recognizing these process-oriented intents as perceived by users and confirmed by performance metrics.

[1]In the remainder of this chapter, we use *HP interaction acts* and *process-oriented intents* interchangeably.

5.2 Related work

A *conversational business process chatbot* enables process actors to interact with a business process in natural language [22]. Interacting with a business process includes intents such as obtaining information about the structure of the process (e.g., inputs and outputs) or about the progress of a process in execution (e.g., which task is currently being processed), or performing activities to advance the state of the process (e.g., API calls, obtain information about the process) [22].

These chatbots were first applied to customer-facing businesses, which involved answering customers' questions about different businesses. Then, organizations started adopting *task-oriented chatbots* to automate tasks in order to reduce the effort required from customers and workers to do their tasks [75]. While today's chatbots may automate some tasks, bot developers have recently started investigating the incorporation of robotic process automation (RPA) to increase automation [180]. For instance, Devy chatbot was proposed to provide automated support in DevOps processes [37]. Authors in [152] developed a chatbot for agile software development teams which analyzes teams' project data to provide insights into their performance. In [141], authors proposed an approach that automatically builds a chatbot from a process model to query process structure, such as when the process ends for a particular actor. Another work proposed a chatbot to query event data allowing users to get insights into specific process executions and retrieve relevant data [122]. All these works are either about domain-specific chatbots or about querying the process execution or structure but do not focus on performing process tasks.

Other works propose approaches to interact with business processes and perform process tasks through chatbots. For instance, Google proposes the use of a chatbot in so-called communication-enabled business process applications (i.e., applications able to orchestrate reactive and proactive communication events) [81]. However, no specific details about the internals of the chatbot infrastructure are provided. The closest work to ours is [114], which proposed a methodology that takes a business process model as input and generates a chatbot to help the users interact with the process. Even though

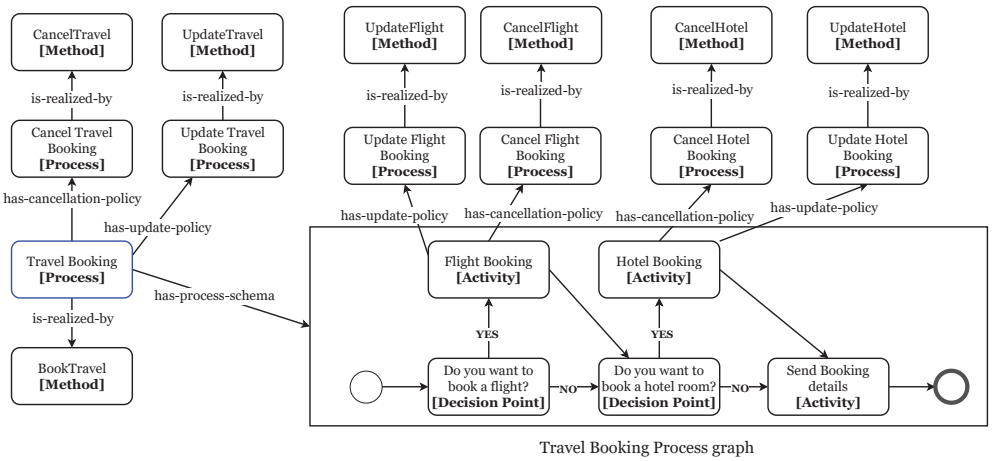

Figure 5.1: Example of a Travel Booking Process Model.

they provide a systematic way to develop chatbot, expanding the scope of the chatbot would require significant coding overhead by developers. In addition, their work does not focus on the recognition of process-oriented intents. In this chapter, we propose a set of HP interaction acts to identify process operations from utterances and thus enable users to perform tasks by naturally interacting with service orchestrations.

5.3 Preliminaries, Scenario and Requirements

In this section, we first introduce some process-related concepts and assumptions (Section 5.3.1). Then, we present a scenario illustrating natural language interactions between a user and multi-step processes (Section 5.3.2). Finally, we define a set of requirements to enable chatbots to support these kinds of interactions (Section 5.3.3).

5.3.1 Preliminaries

A *business process* is a collection of coordinated tasks to achieve a concrete goal [69]. For example, a Travel Booking process can offer full vacation packages by combining several tasks such as Flight and Hotel Bookings. The *schema* of a process can be represented in a variety of forms, such as Petri nets, Event-Driven Process Chains

(EPCs), and Business Process Model and Notation (BPMN) [69]. For simplicity, we represent the process schema as a directed acyclic graph. Figure 5.1 shows an example of a Travel Booking Process graph. The process graph nodes represent *activities* (e.g., Flight Booking) and *decision points* (e.g., do you want to book a flight?).

A process is associated with a set of *exception handling policies*. Exception handling policies are directives that model exceptional situations together with a set of actions that are used to handle exceptions (e.g., cancel a travel booking) [26]. In line with service-oriented realization of business processes [26], in this chapter, we consider that a process is realized by a composite service (e.g., sequence of API calls). Furthermore, a composite service is accessible through an API that includes: a main method to invoke the normal process behavior (e.g., *BookTravel* method in Figure 5.1), and exception handling methods to handle exceptional behaviors of a process (e.g., *CancelTravel* method in Figure 5.1) or an activity (e.g., *CancelHotel* method in Figure 5.1). We also consider that a process has a set of *correlation attributes* that uniquely identify an *instance* (i.e., a specific occurrence or execution of a process) [69]. The values of these correlation attributes act as a *key* that allow to identify a process instance in a unique way. A system auto-generated ticket could be an example of such a correlation attribute.

5.3.2 Scenario

We envision a scenario where a user is able to interact with multi-step processes using natural language utterances (e.g., through a chatbot). Figure 5.2 shows an example of a user-chatbot conversation in which the user is interacting with the Travel Booking process illustrated in Figure 5.1. There are interactions that are triggered by the user (e.g., user requests to start a task) and others by the chatbot (e.g., chatbot asks for information needed to fulfill an activity).

In this scenario, the user initiates the conversation by requesting the *creation of a new instance* of the Travel Booking process (message 1). In the beginning, the chatbot does not expect the user to give all the information that is required for all activities.

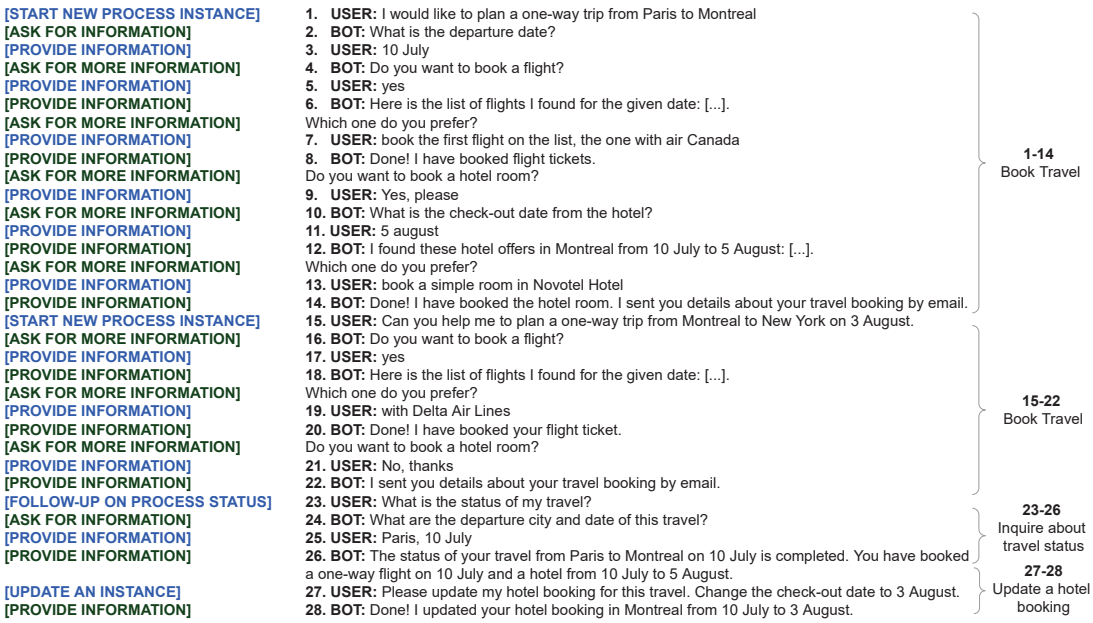

Figure 5.2: Example of natural language conversation between a user and a process-aware chatbot. Interaction acts in blue are triggered by the user and those in green are triggered by the chatbot.

Instead, it *asks for some information* that is needed to start the process, such as the *departure date* (message 2). Later, the chatbot may ask the user for *more information* to perform an activity or even a decision point. For example, to perform the activity *Hotel Booking*, the chatbot asks the user for the hotel check-out date (message 10). Thus, the chatbot does not collect information from the user unless it needs it.

During normal process execution (e.g., booking travel): (i) the chatbot can *ask for more information to fulfill a task* (message 10) or *provide information* about a performed task (message 6), (ii) the user can *provide information* (message 11) or *inquire about task progress* (message 23). As mentioned before, a process is associated with possible run-time exceptions. For example, in message 27, the user wants to change the hotel check-out date. This interaction (changing date) is triggered by the user and involves the update of a previously performed task. However, exception interactions can also be triggered by the chatbot. For example, assume that the airline company canceled the user's flight. The chatbot can trigger an interaction that involves notifying the user about the cancellation and proposing alternatives such as changing the travel date. In

this chapter, we focus on interactions from the user side.

5.3.3 Requirements

Having introduced the scenario and main concepts, we discuss some key requirements to enable natural language interactions between users and multi-step processes. We form an understanding of these requirements based on literature, our experience and experimentation on a range of task-oriented conversation service and business process tools and techniques [32, 33, 35, 69, 247, 248].

R1 Identifying the corresponding process: The chatbot must be able to identify the process to which the user refers to in their tasks. For example, in the previous scenario, the user refers to the Travel Booking process, however, the user can switch between several processes (e.g., visa application process) during the conversation.

R2 Identifying new process instance: The chatbot must be able to identify the utterance that creates a new instance of a process. For example, in utterance 1, the chatbot needs to understand that the user wants to start a new instance of Travel Booking process.

R3 Recognizing process-aware user intents: The chatbot should be able to recognize possibly ambiguous process-aware user intents, including starting a new task, inquiring about task progress, and exceptional behaviors such as canceling or updating tasks.

R4 Handling context: The chatbot must keep track of process executions and must be able to properly handle the context. For instance, context features are important to identify if an utterance concerns the creation of a new process instance or an inquiry regarding an existing process instance.

5.4 Architecture

To support natural language conversations with processes, the chatbot needs a set of services to initiate, monitor, and control task-related conversations. Figure 5.3 shows the workflow between these services. The *Natural Language Understanding (NLU)* service aims to extract *HP interaction acts* and slot-value pairs from the utterance. Details on how to detect these *interaction acts* is presented in Section 5.6.1. The *Process Embedding Service (PES)* aims to identify the *process* that corresponds to the utterance. The *Dialogue Manager (DM)* service aims to infer the dialogue state in terms of the user intent and its slot-value pairs. This DM relies on the *Context Knowledge Service (CKS)* to recognize user intent and infer missing information. Once the DM recognizes the intent and collects all required information, it performs the corresponding action and sends the results to the *Natural Language Generator (NLG)*. The NLG uses then predefined templates to generate human-like responses to the user. In what follows, we describe PES and CKS.

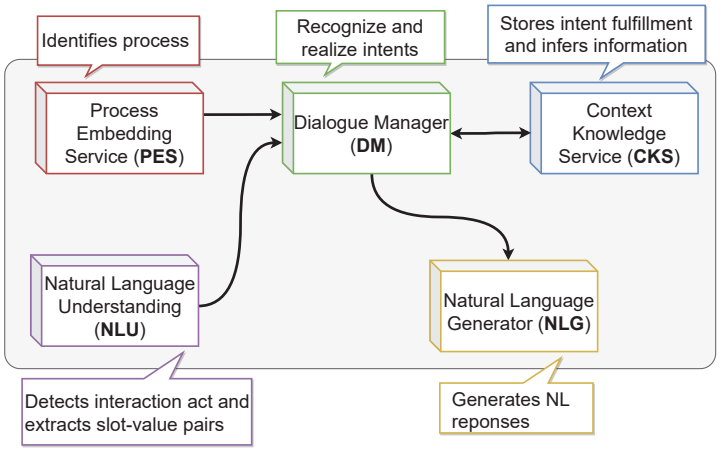

Figure 5.3: General architecture supporting our approach.

5.4.1 Process Embedding Service

Our work builds upon the concept of Word Embedding API element vectors, which has previously been used to facilitate the integration of APIs and chatbots [246]. In this work, we propose a novel process embedding service (PES) that enables chatbots

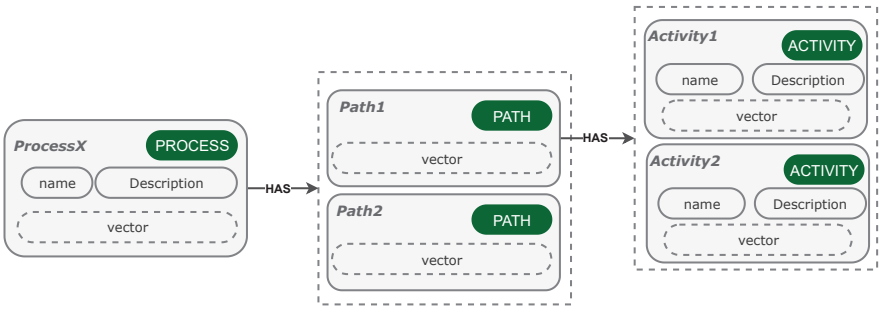

Figure 5.4: Process Knowledge Graph (P-KG).

to identify processes from utterances during natural language conversations. The PES has (i) a process knowledge model and (ii) a set of services to leverage this knowledge.

5.4.1.1 Process Knowledge Model

We propose to represent process elements (i.e., process, paths and activities) as vectors in a vector space model to support natural language interaction with processes [181]. The process knowledge model is denoted as a process knowledge graph (P-KG) with specific types of nodes and relationships. In particular, nodes can describe `Processes`, `Paths`, and `Activities`, as shown in Figure 5.4. Part of the information in the P-KG is the graph representation of what we find in the process model definition. Such information typically includes the process *name*, the process *description*, activity *name*, and activity *description*. This information is added to the P-KG both in *textual* form and in *vectorized* representation as shown in Figure 5.4.

Furthermore, a `Process` node has 4 types of relationships: `is-realized-by` denotes that a process is performed by an API method. For instance, in Figure 5.1, the process *Travel Booking* is performed by the method `BookTravel`. The relationships `has-cancellation-policy` and `has-update-policy` denote that a process has *cancel* and *update* exception policies which are processes by themselves. The `Process` node also has a process graph such as the one illustrated in Figure 5.1. The `Activity` node can have *cancel* and *update* exception policies as well. For instance, in Figure 5.1, the activity *Hotel Booking* has a cancellation policy `Cancel Hotel Booking`.

5.4.1.2 Process Embedding Services

The PES features three services, which are devoted to (i) generating vector embeddings for process elements, (ii) identifying the corresponding process of a given utterance, and (iii) identifying the method that realizes a given process. In what follows, we describe each of these services:

Vector generation Service: This service is used to construct vector embeddings for process elements. It is enabled by the endpoint `"PES/generate_vector? e"` that takes as input a process element `e` and generates its vector embedding. It generates: (i) an *activity vector* by aggregating the information from activity *name* and its *description*; (ii) a *path vector* by aggregating vectors of activities in this path; and (iii) a *process vector* by averaging the information from process *name*, process *description* and all path vectors in this process.

Process Identification Service : This service aims to identify the corresponding process of a given utterance. It is enabled by the endpoint `"PES/sim? u"` that takes as input an utterance `u` and returns the process that corresponds to this utterance. First, it generates the embedding vector of the utterance. Second, it calculates the cosine similarity between this utterance vector and the vector of each process in the P-KG. Then, according to a predefined threshold, the processes with similarities greater than this threshold are kept and ordered. Finally, the service returns the process scoring the highest similarity with the utterance.

Method Identification Service: This service aims to identify a process API method that corresponds to a given process intent. It is enabled by the endpoint `"PES/process_method? p & i & a"` that takes as input a process `p` (e.g., *Travel Booking*), a process intent `i` (e.g., canceling a task), and an activity `a` (e.g., *Hotel Booking*), and returns the corresponding API method (e.g., *CancelHotel* method). Note that the parameter `a` is optional, if its value is not indicated, the endpoint returns the method related to the process (e.g., *CancelTravel* method instead of *CancelHotel* method).

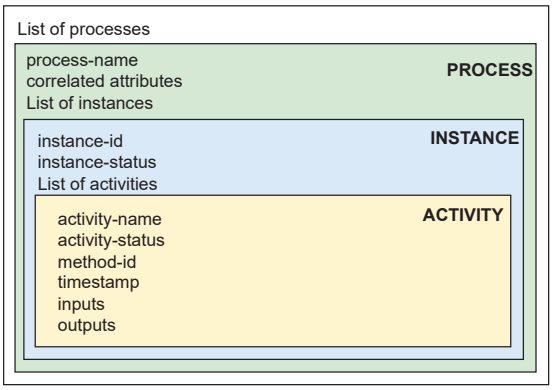

Figure 5.5: Event Data Memory (EDM) Schema.

5.4.2 Context Knowledge Services

In Chapter 4, we proposed the context knowledge service (CKS) that enables to capture contextual knowledge (from conversation history, external services, intent and API schemas) to support the inference of intent slot values during conversations. This CKS has a memory to keep track of all the traces related to each intent fulfillment for a given user. This includes the utterance, the intent, the slot-value pairs, the method call, alongside with its timestamp, its inputs, and its outputs.

In this chapter, we extend the CKS to include relevant information about process instances that should be memorized during conversations for later reuse. More specifically, we include a new memory structure, called *Event Data Memory (EDM)*. This memory keeps track of information about process execution traces (process instances). Figure 5.5 shows the schema of EDM memory. It includes a list of processes where each process has a name and a list of instances. An instance has an id, a status (e.g., pending, in-progress or completed), and a list of activities. An activity has a name, a status, and the method call, alongside with its timestamp, its inputs, and its outputs.

We also extend the core CKS presented in Chapter 4 with two additional services related to processes:

Process Instance Identification Service: This service returns the list of instances for a given process. It is enabled by the endpoint `"CKS/get_instances?  p"` that

takes as input a process `p` and returns the list of instances of `p`.

Correlation attribute Value Retrieval Service: This service provides values of correlation attributes for a given process instance id. It is enabled by the endpoint `"CKS/get_corr_att_val? i"` that takes as input an instance id and returns an array of attribute-value pairs corresponding to correlation attributes values of the instance `i`.

5.5 Process-aware User Intents

Based on requirements identified in Section 5.3.3, conversations regarding a given process-aware task may involve several turns (e.g., starting a travel booking, later inquiring about booking status, modifying travel dates, or canceling the booking). We propose *Human-bot-Process interaction acts* to characterize a set of elementary user intents in conversations between users and multi-step processes. Specifically, we derive four types of process-oriented intents: *start new process instance* intent, *follow-up on process status* intent, *canceling task* intent, and *task update* intent.

To recognize the process-oriented intents, we propose to reuse the hybrid approach proposed in Chapter 4. This approach involves defining rules that combine the detection of HP interaction acts from utterances with additional context and process knowledge, allowing the recognition and realization of the process-oriented intents. The recognition of these intents can be divided into three general steps: *HP interaction act recognition*, *process identification*, and *process instance recognition*. In what follows, we describe each of these steps:

Step 1: HP interaction act Recognition. The first step consists of detecting the *HP interaction act class* (i.e., `Start New Process Instance` class, `Follow-up on Process Status` class, `Task Update` class or `Canceling a Task` class) expressed in the utterance `u` thanks to the NLU service. In Section 5.6.1, we explain how the NLU service allows to detect these HP interaction acts.

Step 2: Process Identification. The second step consists of identifying the process

```
Rule "start-new-process-instance"
when
Trigger
(1) IS_START(u).equals(true)   AND
(2) IS_SIM(u, p).equals(true)   AND
(3) EXIST_INSTANCE(u, p).equals(false)
then
Action
(1) m := "PES/process_method? p & 'start'"
(2) INVOKE_METHOD(m)
```

```
Rule "follow-up-on-process-status"
when
(1) IS_FOLLOW(u).equals(true)   AND
(2) IS_SIM(u, p).equals(true)   AND
(3) EXIST_INSTANCE(u, p).equals(true)
then
(1) i := GET_INSTANCE(u)
(2) SHOW_STATUS(i, p)
```

```
Rule "canceling-task"
when
(1) IS_CANCEL(u).equals(true)   AND
(2) IS_SIM(u, p).equals(true)   AND
(3) EXIST_INSTANCE(u, p).equals(true)
then
(1) i := GET_INSTANCE(u)
(2) a := GET_ACTIVITY(u)
(3) m := "PES/process_method? p & 'cancel' & a"
(4) INVOKE_METHOD(m, i)
```

```
Rule "task-update"
when
(1) IS_UPDATE(u).equals(true)   AND
(2) IS_SIM(u, p).equals(true)   AND
(3) EXIST_INSTANCE(u, p).equals(true)
then
(1) i := GET_INSTANCE(u)
(2) a := GET_ACTIVITY(u)
(3) m := "PES/process_method? p & 'update' & a"
(4) INVOKE_METHOD(m, i)
```

Figure 5.6: Rules to recognize and realize the identified process-oriented intents.

`p` that corresponds to the utterance `u`. To identify processes from natural language utterances, we use the *Process Embedding Service (PES)* presented in Section 5.4.1.2.

Step 3: Process Instance Recognition. This last step aims to identify the process instance from the utterance `u`, then checks if the identified process instance already exists or not. First, the chatbot invokes the *Process Instance Identification* CKS service to retrieve the set of instances `set_i` of the process `p`. Then, it extracts the values of correlation attributes of the instance that the user is referring to. Finally, it compares the extracted values of correlation attributes with those of instances `set_i` to check if the identified process instance already exists or not. For example, if the chatbot detects *canceling task* interaction act, the process instance should exist to be able to cancel it.

In what follows, we describe the process-oriented intents and define their rules:

5.5.1 Start New Process Instance

This intent allows to identify whether the user utterance expresses a task that requires the creation of a new process instance. In general, when users ask for a new task, they provide general information describing this task, and sometimes they provide more detailed information about this task. For example, in Figure 5.2 utterance 1,

the user describes the task *book travel* and provides additional details about this travel such as the departure and destination cities. The chatbot needs this information to identify the process and to check if the utterance concerns the creation of a new process instance. Figure 5.6 shows the specification of the rule related to *start-new-process-instance* intent. This rule consists of *trigger* and *action* clauses. The trigger clause defines three boolean conditions: (1) The condition `IS_START()` checks if the utterance `u` expresses a *start new instance* HP interaction act. (2) The condition `IS_SIM()` is expressed over the *Process Identification PES* service. This condition checks if the process `p` corresponds to the utterance `u`. (3) The last condition `EXIST_INSTANCE()` compares values of the correlation attributes with those of the existing instances to check if the identified instance does not exist. If the conditions are satisfied, the chatbot (1) invokes the *Method Identification PES* service to get the method `m` and (2) triggers `m` to start the process execution.

5.5.2 Follow-up on Process Status

Follow-up on a process can be a complex task by itself, such as querying a process execution history to get information about run time and performance of a task. We focus on a specific type of follow-up which is *following up on process status* that allows inquiring about process instance status (e.g., pending, in-progress or completed). For example, in Figure 5.2 utterance 23, the user is inquiring about travel status. Figure 5.6 shows the specification of the rule related to *follow-up-on-process-status* intent. The conditions are the same as those defined in the previous rule, except that this rule needs to detect a *follow-up* HP interaction act and the process instance should exists. If all conditions are satisfied, the chatbot (1) retrieves the corresponding instance and (2) lists the status of this instance.

5.5.3 Task Update

This intent allows to identify whether the user wants to update an existing process instance. For example, in Figure 5.2 utterance 27, the user wants to update the hotel

check-out date. A user can request to update an information in the whole process (e.g., update travel date), or a specific activity in the process (e.g., update the date of a hotel booking). We model an activity as an input parameter of the intent *task update*, thus the chatbot can extract from the user utterance the activity that the user wants to update. For example, in utterance 27, the chatbot extracts *hotel booking* as a value of *activity* parameter. Figure 5.6 shows the rule related to *task-update* intent. The conditions are the same as those defined in the first rule, except that this rule needs to detect a *task update* HP interaction act and the process instance should exists. If all conditions are satisfied, the chatbot (1) retrieves the corresponding instance, (2) extract the value of the *activity* parameter, (3) invokes the *Method Identification PES* service to get the the corresponding method `m` and (4) triggers `m` to update the corresponding task (i.e., activity or process).

5.5.4 Canceling a Task

This intent allows to identify whether the user utterance expresses a task cancellation of an existing process instance. The user can request to cancel the whole process (e.g., canceling travel bookings), or a specific task in the process (e.g., canceling hotel booking). The steps to recognize and realize *canceling task* intent are the same as those in the *task update* intent (Figure 5.6).

5.6 Validation

The first objective of the study was to explore the *effectiveness* of the chatbot improved with Process Embedding Service (PES) and process-oriented intent rules, i.e., its capability of recognizing correctly the process-oriented intents and reducing unnecessary interactions. The second objective was to assess the impact of enabling interaction with a process as opposed to leaving users to orchestrate services themselves to fulfill their goals. In what follows, we first discuss different strategies to collect utterance paraphrases and build a dialogue dataset to train process-oriented intent recognition

models (Section 5.6.1), then we present the study method (Section 5.6.2), and finally we discuss the results (Section 5.6.3).

5.6.1 Process-oriented Intent Training Dataset Construction

Due to lack of benchmark training datasets for process-oriented intents, we built training dataset of utterances about three processes (i.e., *Plan Event*, *Book Travel* and *Shopping*) and four process-oriented intents. In this section, we discuss how we built this dialogue dataset and explain how the NLU service allows to detect the HP interaction acts (i.e., *start new process instance*, *canceling task*, *task update*, and *follow-up on process status*).

The accuracy of NLU detection models depends on the quality (and size) of the dataset of utterances used for training these models. A well known approach to build dialogue datasets involves expanding an initial set of seed utterances by means of *paraphrasing* [168]. Paraphrasing is a task that aims to reformulate a given utterance into another with the same semantic meaning [168]. *Crowdsourcing* and *automatic* paraphrasing are two popular strategies used to generate paraphrases [168]. In a crowdsourced paraphrasing process, an initial seed utterance is presented as a starting point, and workers are asked to paraphrase the seed to produce new utterances [168]. In an automated paraphrasing, there are models that automatically generate paraphrases from a seed of utterances [28, 168]. To build the dialogue dataset, we first crowdsource a set of utterances by starting with a seed that we provided. Then, we paraphrase the obtained set of utterances using an automatic paraphrasing model to generate more utterances. In what follows, we explain in detail each of these two steps.

5.6.1.1 Crowd-based paraphrasing

This first step aims to generate a set of utterances, for each of the three processes (i.e., *Plan Event*, *Book Travel* and *Shopping*) and each of the four process-oriented intents (i.e., *start new process instance*, *canceling task*, *task update*, *follow-up on process status*),

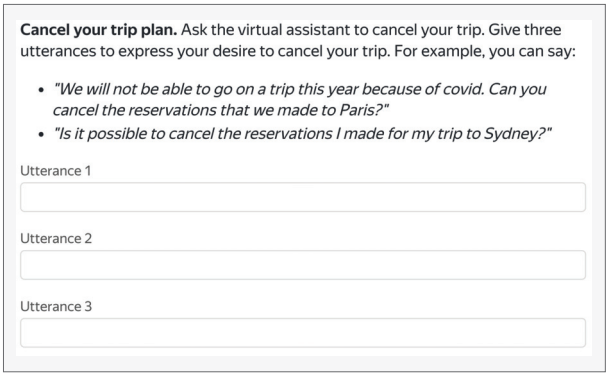

Figure 5.7: Crowdsourcing sub-task to provide paraphrases for the intent *Canceling a Task*.

starting from a seed of utterances that we provide.

We design three crowdsourcing tasks where each task is related to one of the three processes. Each task provides: (i) a scenario describing the process and explaining to the workers that there is a virtual assistant that allows them to interact with this process, and (ii) four sub-tasks related to the four process-oriented intents. Each of these sub-tasks prompts crowd workers to provide 3 paraphrases for a given seed of 2 utterances (refer to Figure 5.7 for an illustrative example of this sub-task).

We ran the three crowdsourcing tasks on the crowdsourcing platform Toloka [16] and recruited workers who had passed an English test (set by the platform) and were ranked top-60%. Each task was assigned to 10 workers, and each worker wrote 3 paraphrases per intent (sub-task). Workers were paid 0.5 USD per task. This crowdsourcing step yielded a dataset of 360 utterances where there are 120 utterances per process and 90 utterances per intent. Among these utterances, workers provide some utterances that do not convey the requested intent. For example, one worker provides the utterance *"please abort booking of the conference center"* for the intent *task update* instead of *canceling task*. We have manually labeled the utterances as *correct*, if the utterance conveys the requested intent, or *not correct*, otherwise.

5.6.1.2 Automatic paraphrasing

This second step expands the paraphrases generated through crowdsourcing by leveraging *Empath* service [70]. Empath is a knowledge base service for generating terms (e.g., facebook, twitter) given a lexical category (e.g., social media).

In our approach, the utterance paraphrases are expanded as follows: First, (i) for each utterance, we apply a Part-of-Speech tagging using spacy [15]. Then, (ii) for each VERB or NOUN token in the paraphrase, we substitute the token with terms returned by Empath when the token is used an input category. Empath allows increasing the lexical diversity of the generated paraphrases. A new paraphrase is generated with each term returned by Empath. For example, if the list of returned items contains 8 terms, we will generate 8 new paraphrases. A further filtering step is necessary to discard semantically unrelated paraphrases. Thus, (iii) we compute a cosine similarity score between the Universal Sentence Encoder (USE) embeddings [181] of the initial utterance and the generated paraphrase to discard paraphrases with a cosine score below a predefined threshold.

5.6.1.3 Recognizing Process-oriented intents using Classification Model

We split the dataset generated from the *crowd-based paraphrasing* step into 30% test dataset and 70% train dataset (*train-crowd*). Then, we augment the *train-crowd* dataset with more utterances by automatically generating paraphrases as described in *automatic paraphrasing* step (*train-hybrid*).

We can use any classification model such as Naive Bayes and Support Vector Machine (SVM) [148] to classify utterance into a corresponding process-oriented intent class. In the current implementation, we use a Logistic Regression classifier [148] and we encode the classifier input based on the distributional semantics of utterances using three different word and sentence embedding models: Word2Vec, GloVe, and USE [181].

Table 5.1: HP Interaction Act Detection Accuracy

	Start		Follow-up		Update		Cancel		Mean	
Dataset	crowd	hyb	crowd	hyb	crowd	hyb	crowd	hyb	crowd	hyb
W2Vec	0.90	0.91	0.85	0.82	0.81	0.84	0.77	0.84	0.8325	0.8525
Glove	0.62	0.66	0.63	0.61	0.60	0.63	0.64	0.65	0.6225	0.6375
USE	0.94	**0.97**	0.89	**0.94**	0.87	**0.90**	0.90	**0.93**	0.9	**0.935**

Table 5.1 shows accuracy[2] of process-oriented intent detection using only the crowdsourcing method as opposed to the hybrid method (i.e., crowd-based and automatic paraphrasing methods). For the four process-oriented intents, we can see that the classifier achieved a great performance, however the USE model performed better accuracy than Word2Vec and GloVe models. This can be explained by the fact that USE model takes into account the context of the word in the sentence while Word2Vec and GloVe models generate a word embedding independently of the context in the sentence. We can also see that using hybrid method as opposed to the crowd method allows to produce better accuracy. For example the accuracy produced using USE model and the hybrid method produced +3.5% of accuracy compared to the one produced using USE model and only the crowd method. The datasets are available at the following link as well as the scripts used for computing accuracy and generating paraphrases[3].

5.6.2 Methods

5.6.2.1 Experimental design

We recruited participants via email from our extended network of contacts. The call resulted in a total of 17 participants. The evaluation scenario consisted of planning a trip to two different destinations, which involved i) booking flights, taxi and accommodations, ii) buying a product needed for the two destinations, and iii) scheduling an appointment at a local test centre. The scenario required participants to perform tasks

[2]We use *F1-score* to rate model's accuracy. F1-score is defined as the harmonic mean between precision and recall [148].

[3]Process Intent Recognition materials: `https://tinyurl.com/DAR-Materials`

associated with three underlying processes (Travel Booking, Shopping and Scheduling an appointment), each processes coordinating two or more activities. Participants were asked to complete 4 tasks in this scenario:

- *T1*: starting new process instances.
- *T2*: updating information of process instances.
- *T3*: following up on process statuses.
- *T4*: canceling process instances.

We followed a within-subjects design tasking participants to complete the above tasks by interacting with two chatbots representing the following conditions:

- *Baseline-chatbot*. The baseline implements a standard conversational management (provided by DialogFlow [5]), without rules, PES, and CKS support.
- *Process-chatbot*. This chatbot is implemented with PES and CKS support as well as the rules. The aim is to assess the performance gain in chatbots in handling process-oriented intents.

5.6.2.2 Procedure

The study was conducted online. Participants received a link to an online form that included an informed consent, all the instructions and links to the two chatbots. After reading the informed consent and agreeing to participate, participants were introduced to the evaluation scenario and tasks (T1-T4). They were asked to perform those tasks with the two chatbots, in a counter-balanced design (i.e., the order in which chatbots were presented was randomized to counter ordering effects). After interacting with each chatbot, participants were asked: to describe positive and negative aspects of the chatbots, to select their preferred chatbot, to specify why, and provide quantitative feedback on their experience along three dimensions: *naturalness* (ability to fulfill user tasks in human-like conversations), *repetitiveness* (ability to avoid redundant questions) and *understanding* (ability to interpret user requests).

5.6.2.3 Data analysis

We performed an analysis of conversation logs so as to assess the effectiveness of our approach in recognizing the process-oriented intents. These are computed in relation to optimal conversation scenarios[4] that we designed based on participants conversations. The effectiveness is calculated by considering the following metrics: number of (M1) conversation turns, (M2) prompts asking for missing information, (M3) process correctly identified and (M4) process-oriented intents correctly recognized. These metrics are calculated per user conversation, aggregated (mean) and then used to compute the relative performance against the optimal scenario.

5.6.3 Results

5.6.3.1 Effectiveness

Table 5.2 shows the relative performance by task of both baseline-chatbot and process-chatbot in relation to the optimal reference scenario. For the four tasks, we can see that process-chatbot experienced a boost in performance M1 and M2 (mean across tasks 86,01% and 83,89% respectively), approaching the efficiency of the reference scenario in terms of number of turns (M1) and prompts (M2), and thus reducing unnecessary interactions. This level of performance is possible thanks to the PES and CKS services and the defined process-oriented rules that allow to perform a mean relative performance across tasks for process identification (M3) and intent recognition (M4) of 90,48% and 85,27% respectively. In contrast, not supporting the process-oriented rules leads the baseline-chatbot to perform poorly in comparison, with the best performance being at around 36,70% for the metrics considered.

By analyzing the conversations, we noticed that the recognition error of the process-oriented intents by the process-chatbot is mostly caused by the detection error of the correct HP interaction act during the conversation. However, for the four tasks, we

[4]Scenarios assuming ideal accuracy of process-oriented intent recognition.

Table 5.2: Performance of experimental conditions for each task according to the relevant metrics. Values in bold denote best performance.

	Baseline-chatbot			Process-chatbot			
Task	M1 TURNS	M2 PROMPTS	M4 INTENT	M1 TURNS	M2 PROMPTS	M3 PROCESS	M4 INTENT
T1 (new)	54,14%	45,59%	51,75%	**95,73%**	**91,67%**	**92,86%**	**88,68%**
T2 (update)	31,58%	26,67%	25,00%	**85,71%**	**80,00%**	**91,67%**	**83,33%**
T3 (follow up)	47,15%	51,28%	27,86%	**82,61%**	**88,89%**	**85,71%**	**85,71%**
T4 (cancel)	27,27%	18,75%	33,33%	**80,00%**	**75,00%**	**91,67%**	**83,33%**
Mean	40,04%	35,57%	34,49%	**86,01%**	**83,89%**	**90,48%**	**85,27%**

can see that the process-chatbot is close to the reference scenario for process-oriented intent recognition (M4) with a mean relative performance across tasks of 85,27%. These results provide evidence for the benefits and effectiveness of handling process-oriented rules during user-chatbot-process conversations. For the four tasks, we can also see that process-chatbot experienced a boost in performance for M1, M2, M3 and M4 (mean across tasks 86.01%, 83.89%, 90,48% and 85,27% respectively), approaching the efficiency of the reference ideal scenario in terms of number of turns, prompts, process identification and intent recognition.

5.6.3.2 User experience

All but two participants (15/17 participants) expressed a preference towards the process-chatbot as opposed to the baseline-chatbot. The two exceptions were due to interaction acts detection that led to the enactment of the wrong services. The feedback to the specific user experience questions, highlighted the reasons behind the preference. The majority of participants reported that process-chatbot interactions described *naturalness* (11/17), less *repetitiveness* (11/17) and *understanding* (12/17), whereas the baseline-chatbot was poorly rated on these fronts (2/17).

The qualitative analysis of open-ended feedback identified specific qualities behind the preference for process-chatbot. Participants commented its ability to *guide and suggest next steps* (8 participants) (e.g., *"[the chatbot] proposed [to] me directly the next action to do without asking for it"*, P14), the ability to *orchestrate operations* with underlying services (7 participants) (e.g., *"[in this chatbot] services are already chained for what is necessary to plan the trip"*, P12), and the ability to *keep track of the context* (6 partici-

pants) (e.g., *"[it] remembers previous answers and use them when asking a new request"*, P5). A better user experience also emerged as a salient quality (10 participants) with *"fluid interactions"* as the most common adjective.

5.7 Conclusion

In this chapter we proposed process-oriented intents that are relevant to represent natural language conversations between the user and multi-step processes. We devised an approach that combines recognition of these process-oriented intents from user utterances with additional context and process knowledge to enable human users to perform tasks by naturally interacting with service orchestrations.

Our work also comes with its own limitations and space for possible improvements. To detect all possible process-oriented intents, a more in-depth study is necessary, which could involve analyzing various business process models. For instance, the intent of *"resuming a process instance"* can be considered as a process-oriented intent triggered by the user during a conversation between a human and a process-aware chatbot. A user can initiate multiple process instances without completing them, such as starting a travel request and stopping at a certain point. For example, the chatbot asks the user to select a hotel, but the user does not respond and switches to another process. Later, the user returns to the travel request and says *"I choose the hotel [hotel-name] for my travel to Lyon,"* and the chatbot needs to detect this as a *resume* intent.

Chapter 6

Conclusion and Future Directions

Contents

With the ongoing growth and development of task-oriented chatbots, there has been a surge of interest among users in investigating chatbots usefulness. Nevertheless, the capability of these chatbots to support composite user intents is a major challenge and depends on gathering a massive amount of annotated data to recognize these intents. This thesis identified and characterized a set of composite dialogue patterns and proposed reusable and extensible techniques to recognize and realize composite user intents in task-oriented human-bot conversations.

In this chapter we provide a summary of the undertaken research issues (Section 6.1) and a summary of research outcomes (Section 6.2). We also discuss some future research directions (Section 6.3).

6.1 Summary the Research Issues

This thesis presented several novel concepts and techniques to fill critical gaps in connecting complex natural language utterances with conversational services. Chapter 2 provided background and identified open issues and the state-of-the-art of approaches for context and dialogue management. In the first contribution (Chapter 3), we identified and characterized a set of composite dialogue patterns and extended a conversational model to represent them. In the second contribution (Chapter 4), we proposed a reusable and extensible approach that uses a sophisticated context service to recognize and realize composite intents. In the third study (Chapter 5), we focused on the superimposition of task-oriented chatbots over composite services by introducing the concept of Human-bot-Process interaction acts.

6.2 Summary of the Research Outcomes

The first outcome, introduced in chapter 3, consists of extending a conversational model to represent and reason about composite user intents. We identified and characterized a set of composite dialogue patterns to endow bot platforms with reusable functionality to recognise compositional conversational flows and reduce the development complexity.

The second outcome, presented in chapter 4, consists of a novel reusable and extensible approach that uses a sophisticated context service to recognize and realize composite intents. This approach relies on (i) existing natural language processing and machine learning techniques to extract natural language features and (ii) a rule-based approach that leverages these features together with contextual knowledge, enabled by composite dialogue patterns and other metadata, to capture different classes of composite intents in a generic way. The contextual knowledge required at run-time to recognize composite intents is extracted from different sources including conversation history, enriched entities, intents and API schemas and represented in graph structure. We believe that the proposed approach charts novel abstractions that unlock the seamless and scalable integration of natural language-based conversations with software-enabled services.

Finally, the third outcome, introduced in chapter 5, consists of enabling the superimposition of task-oriented chatbots over composite services. We proposed process-oriented intents that are relevant to represent natural language conversations between the user and multi-step processes. We devised an approach that combines the recognition of these process-oriented intents from user utterances with additional context and process knowledge. This approach enables human users to perform tasks by naturally interacting with service orchestrations.

6.3 Future Research Directions

While we addressed some issues in this thesis, there are still many further exciting challenges and opportunities in dialogue systems domain. We take the view that although concrete user intents are application-specific, in many cases it is possible to capture in a generic way different classes of user intents that may be composed during a conversation of a user with a software service or process. In what follows, we present some possible *Human-AI-Service Intent Patterns* that could be useful to improve chatbots robustness and effectiveness.

Considering patterns that cater for *human-AI interaction breakdowns* (e.g., requesting user to confirm an inferred slot value or to choose between several options), can be one of the extensions. We argue that dialogue management systems should be endowed with the ability to systematically manage human-bot communication breakdowns in accordance with high-level and configurable recovery strategies. In addition to composite intent recognition, it is interesting to develop a robust breakdown recovery techniques so that (i) relevant recovery strategies can be constructed systematically and incrementally, and (ii) dialogue management systems continuously adapt behaviour of chatbots by reasoning and acting upon recovery strategies.

While in this thesis we focused on the traditional intent slots values, it could be interesting to improve chatbots with the ability to extract and understand subjective, or Quality of Service (QoS), attribute values from user utterances. When people search for

products (e.g., shoes) or services (e.g., restaurants), they are always facing situations of choice. Studies show that when people make decisions (i.e., choose a product/service), they are prone to rely on subjective information (e.g., delicious food, quietness, friendly staff) rather than relying on objective information (e.g., food type, price range, location) [134]. Subjectivity refers to the quality of being based on personal opinions, feelings, or perspectives, rather than on objective facts or evidence [134]. For example, when choosing a restaurant, people are attracted by restaurants that offer great experiences, such as delicious food, quietness, or helpful waitstaff, in addition to objective information, such as specific types of food or location [74, 134]. Extracting QoS attributes for intent recognition is challenging because it requires an understanding of contextual and subjective information (e.g., a user utterance that refers to booking a quiet restaurant) compared to traditional intent slot values that are factual values (e.g., price of a book). To the best of our knowledge there is no dialogue dataset that annotates utterances with QoS attributes. Creating QoS recognition models will require techniques information extraction from potentially noisy user conversations and social data (e.g., from product reviews, comments, feedback and conversations).

Task-based chatbots often ask users to provide sensitive information, such as personal identification. It is essential to ensure that these chatbots respect user privacy and maintain confidentiality. Therefore, the development of privacy-aware task-oriented chatbots is another important area of research. One possible approach for developing privacy-aware chatbots is to use NLP techniques to reason about privacy-preserving conversations. It involves analyzing the interaction between the user and the chatbot in order to identify potential privacy risks and develop strategies to prevent them (e.g., obtaining user consent before collecting sensitive information, avoid asking unnecessary questions).

Bibliography

[1] Amazon lex, https://aws.amazon.com/fr/lex/ - Last accessed on 2022-12-15

[2] Chatfuel, https://chatfuel.com/ - Last accessed on 2022-12-15

[3] Chatgpt, https://chat.openai.com/ - Last accessed on 2023-01-12

[4] Chatscript, https://github.com/ChatScript/ChatScript - Last accessed on 2022-12-17

[5] Dialogflow, https://dialogflow.com/ - Last accessed on 2022-12-15

[6] Flowxo, https://flowxo.com/ - Last accessed on 2022-12-15

[7] Ibm watson platform, https://assistant-eu-gb.watsonplatform.net/ - Last accessed on 2022-12-15

[8] Manychat, https://manychat.com/ - Last accessed on 2022-12-17

[9] Manychat, https://github.com/huggingface/neuralcoref - Last accessed on 2022-12-17

[10] Microsoft luis, https://www.luis.ai/ - Last accessed on 2022-12-15

[11] Mitsuku, https://chat.kuki.ai/ - Last accessed on 2022-12-15

[12] Openai, https://openai.com/ - Last accessed on 2023-01-10

[13] Replika, https://replika.ai/ - Last accessed on 2022-12-15

[14] Rivescript, https://www.rivescript.com/ - Last accessed on 2022-12-17

[15] spacy, `https://spacy.io/` - Last accessed on 2022-12-17

[16] Toloka, `https://toloka.yandex.com/` - Last accessed on 2022-12-17

[17] Wit.ai, `https://wit.ai/` - Last accessed on 2022-12-15

[18] Van der Aalst, W.M., Bichler, M., Heinzl, A.: Robotic process automation. Business & information systems engineering **60**, 269–272 (2018)

[19] Abdellatif, A., Badran, K., Costa, D.E., Shihab, E.: A comparison of natural language understanding platforms for chatbots in software engineering. IEEE Transactions on Software Engineering **48**(8), 3087–3102 (2021)

[20] Agarwal, S., Jezabek, J., Biswas, A., Barut, E., Gao, B., Chung, T.: Building goal-oriented dialogue systems with situated visual context **36**(11), 13149–13151 (2022)

[21] Andreas, J., Bufe, J., Burkett, D., Chen, C., Clausman, J., Crawford, J., Crim, K., DeLoach, J., Dorner, L., Eisner, J., et al.: Task-oriented dialogue as dataflow synthesis. Transactions of the Association for Computational Linguistics **8**, 556–571 (2020)

[22] Baez, M., Daniel, F., Casati, F., Benatallah, B.: Chatbot integration in few patterns. IEEE Internet Computing **25**(3), 52–59 (2020)

[23] Banchs, R.E., Jiang, R., Kim, S., Niswar, A., Yeo, K.H.: AIDA: Artificial intelligent dialogue agent. In: Proceedings of the SIGDIAL 2013 Conference. pp. 145–147 (2013)

[24] Barto, A., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems: Theory and Applications **13** (2003)

[25] Barukh, M.C., Zamanirad, S., Baez, M., Beheshti, A., Benatallah, B., Casati, F., Yao, L., Sheng, Q.Z., Schiliro, F.: Cognitive augmentation in processes. pp. 123–137. Springer (2021)

[26] Benatallah, B., Sheng, Q.Z., Dumas, M.: The self-serv environment for web services composition. IEEE internet computing **7**(1), 40–48 (2003)

[27] Bengio, Y., Ducharme, R., Vincent, P.: A neural probabilistic language model. Advances in neural information processing systems **13** (2000)

[28] Berro, A., Fard, M.A.Y.Z., Baez, M., Benatallah, B., Benabdeslem, K.: An extensible and reusable pipeline for automated utterance paraphrases (2021)

[29] Bickmore, T., Cassell, J.: Small talk and conversational storytelling in embodied conversational interface agents. In: AAAI fall symposium on narrative intelligence. pp. 87–92 (1999)

[30] Bohus, D., Rudnicky, A.I.: Ravenclaw: dialog management using hierarchical task decomposition and an expectation agenda. In: INTERSPEECH (2003)

[31] Bouguelia, S., Berro, A., Benatallah, B., Báez, M., Brabra, H., Zamanirad, S., Kheddouci, H.: Process-oriented intents: A cornerstone for superimposition of natural language conversations over composite services pp. 575–583 (2022)

[32] Bouguelia, S., Brabra, H., Benatallah, B., Baez, M., Zamanirad, S., Kheddouci, H.: Context knowledge-aware recognition of composite intents in task-oriented human-bot conversations pp. 237–252 (2022)

[33] Bouguelia, S., Brabra, H., Zamanirad, S., Benatallah, B., Baez, M., Kheddouci, H.: Reusable abstractions and patterns for recognising compositional conversational flows pp. 161–176 (2021)

[34] Brabra, H., Báez, M., Benatallah, B., Gaaloul, W., Bouguelia, S., Zamanirad, S.: Dialogue management in conversational systems: a review of approaches, challenges, and opportunities. IEEE Transactions on Cognitive and Developmental Systems (2021)

[35] Brabra, H., Báez, M., Benatallah, B., Gaaloul, W., Bouguelia, S., Zamanirad, S.: Dialogue management in conversational systems: a review of approaches, challenges, and opportunities. IEEE Transactions on Cognitive and Developmental Systems (2021)

[36] Bradley, N., Fritz, T., Holmes, R.: Context-aware conversational developer assistants. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). pp. 993–1003. IEEE Computer Society (2018)

[37] Bradley, N.C., Fritz, T., Holmes, R.: Context-aware conversational developer assistants. In: Proceedings of the 40th International Conference on Software Engineering. pp. 993–1003 (2018)

[38] Bradley, N.C., Fritz, T., Holmes, R.: Context-aware conversational developer assistants. In: Proceedings of the 40th International Conference on Software Engineering. pp. 993–1003 (2018)

[39] Braun, V., Clarke, V.: Successful Qualitative Research: A Practical Guide for Beginners. SAGE (2013)

[40] Budzianowski, P., Ultes, S., Su, P.H., Mrkšić, N., Wen, T.H., Casanueva, I., Rojas-Barahona, L.M., Gašić, M.: Sub-domain modelling for dialogue management with hierarchical reinforcement learning. In: Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue. pp. 86–92 (2017)

[41] Bui, T.H., Poel, M., Nijholt, A., Zwiers, J.: A tractable hybrid ddn–pomdp approach to affective dialogue modeling for probabilistic frame-based dialogue systems. Natural Language Engineering **15**(2), 273–307 (2009)

[42] Burgan, D.: Dialogue systems & dialogue management. Tech. rep., DST Group Edinburgh: Edinburgh, Australia (2017)

[43] Callejas, Z., Griol, D., Engelbrecht, K.P., López-Cózar, R.: A clustering approach to assess real user profiles in spoken dialogue systems. In: Natural interaction with robots, knowbots and smartphones, pp. 327–334. Springer (2014)

[44] Canonico, M., De Russis, L.: A comparison and critique of natural language understanding tools. Cloud Computing **2018**, 120 (2018)

[45] Casanueva, I., Budzianowski, P., Su, P.H., Mrkšić, N., Wen, T.H., Ultes, S., Rojas-Barahona, L., Young, S., Gašić, M.: A benchmarking environment for

reinforcement learning based task oriented dialogue management. In: 31st Conference on Neural Information Processing Systems (2017)

[46] Chakrabarti, C., Luger, G.F.: A semantic architecture for artificial conversations. In: The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems. pp. 21–26 (2012)

[47] Chao, G.L., Lane, I.: Bert-dst: Scalable end-to-end dialogue state tracking with bidirectional encoder representations from transformer. arXiv preprint arXiv:1907.03040 (2019)

[48] Chao, G.L., Lane, I.: Bert-dst: Scalable end-to-end dialogue state tracking with bidirectional encoder representations from transformer. pp. 1468–1472 (09 2019)

[49] Chen, H., Liu, X., Yin, D., Tang, J.: A survey on dialogue systems: Recent advances and new frontiers. Acm Sigkdd Explorations Newsletter **19**(2), 25–35 (2017)

[50] Chen, H., Liu, X., Yin, D., Tang, J.: A survey on dialogue systems: Recent advances and new frontiers. SIGKDD Explor. Newsl. **19**(2), 25–35 (2017)

[51] Chen, L., al.: Schema-guided multi-domain dialogue state tracking with graph attention neural networks. Proc. AAAI 2020 **34**, 7521–7528 (2020)

[52] Chen, L., Chen, Z., Tan, B., Long, S., Gasic, M., Yu, K.: Agentgraph: Toward universal dialogue management with structured deep reinforcement learning. IEEE/ACM Transactions on Audio, Speech, and Language Processing **27**, 1378–1391 (2019)

[53] Chen, L., Lv, B., Wang, C., Zhu, S., Tan, B., Yu, K.: Schema-guided multi-domain dialogue state tracking with graph attention neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 7521–7528 (2020)

[54] Chen, L., Lv, B., Wang, C., Zhu, S., Tan, B., Yu, K.: Schema-guided multi-domain dialogue state tracking with graph attention neural networks. Proceedings of the AAAI Conference on Artificial Intelligence **34**(05), 7521–7528 (2020)

[55] Chen, L., Lv, B., Wang, C., Zhu, S., Tan, B., Yu, K.: Schema-guided multi-domain dialogue state tracking with graph attention neural networks **34**(05), 7521–7528 (2020)

[56] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1724–1734. Association for Computational Linguistics (2014)

[57] Christmann, P., Saha Roy, R., Abujabal, A., Singh, J., Weikum, G.: Look before you hop: Conversational question answering over knowledge graphs using judicious context expansion. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management. pp. 729–738 (2019)

[58] Christmann, P., Saha Roy, R., Weikum, G.: Efficient contextualization using top-k operators for question answering over knowledge graphs. arXiv e-prints pp. arXiv–2108 (2021)

[59] Cranshaw, J., Elwany, E., Newman, T., Kocielnik, R., Yu, B., Soni, S., Teevan, J., Monroy-Hernández, A.: Calendar. help: Designing a workflow-based scheduling agent with humans in the loop. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. pp. 2382–2393 (2017)

[60] Cuayahuitl, H.: Simpleds: A simple deep reinforcement learning dialogue system. In: Dialogues with Social Robots (2017)

[61] Cuayáhuitl, H., Yu, S.: Deep reinforcement learning of dialogue policies with less weight updates. In: INTERSPEECH (2017)

[62] Dale, R.: The return of the chatbots. Natural Language Engineering **22**(5), 811–817 (2016)

[63] Das, A., Kottur, S., Gupta, K., Singh, A., Yadav, D., Moura, J.M., Parikh, D., Batra, D.: Visual dialog. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 326–335 (2017)

[64] Deng, L., Tur, G., He, X., Hakkani-Tur, D.: Use of kernel deep convex networks and end-to-end learning for spoken language understanding. In: 2012 IEEE Spoken Language Technology Workshop (SLT). pp. 210–215. IEEE (2012)

[65] Deoras, A., Sarikaya, R.: Deep belief network based semantic taggers for spoken language understanding. (2013)

[66] Dhamdhere, K., McCurley, K.S., Nahmias, R., Sundararajan, M., Yan, Q.: Analyza: Exploring data with conversation. In: Proceedings of the 22nd International Conference on Intelligent User Interfaces. pp. 493–504 (2017)

[67] Dinan, E., Roller, S., Shuster, K., Fan, A., Auli, M., Weston, J.: Wizard of wikipedia: Knowledge-powered conversational agents. arXiv preprint arXiv:1811.01241 (2018)

[68] van Dis, E.A., Bollen, J., Zuidema, W., van Rooij, R., Bockting, C.L.: Chatgpt: five priorities for research. Nature **614**(7947), 224–226 (2023)

[69] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of business process management, vol. 1. Springer (2013)

[70] Fast, E., Chen, B., Bernstein, M.S.: Empath: Understanding topic signals in large-scale text. In: Proceedings of the 2016 CHI conference on human factors in computing systems. pp. 4647–4657 (2016)

[71] Fast, E., Chen, B., Mendelsohn, J., Bassen, J., Bernstein, M.S.: Iris: A conversational agent for complex tasks. In: Proceedings of the 2018 CHI conference on human factors in computing systems. pp. 1–12 (2018)

[72] Fatemi, M., El Asri, L., Schulz, H., He, J., Suleman, K.: Policy networks with two-stage training for dialogue systems. In: Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue. pp. 101–110. Association for Computational Linguistics (2016)

[73] Feng, S., Lubis, N., Geishauser, C., Lin, H.c., Heck, M., van Niekerk, C., Gasic, M.: Emowoz: A large-scale corpus and labelling scheme for emotion recognition in task-oriented dialogue systems. In: Proceedings of the Thirteenth Language Resources and Evaluation Conference. pp. 4096–4113 (2022)

[74] Gaci, Y., Ramírez, J., Benatallah, B., Casati, F., Benabdslem, K.: Subjectivity aware conversational search services. In: 24th International Conference on Extending Database Technology (EDBT 2021) (2021)

[75] Galitsky, B.: Developing enterprise chatbots. Springer (2019)

[76] Gao, J., Galley, M., Li, L.: Neural approaches to conversational ai. In: The 41st international ACM SIGIR conference on research & development in information retrieval. pp. 1371–1374 (2018)

[77] Gao, J., Galley, M., Li, L.: Neural approaches to conversational AI. Foundations and Trends in Information Retrieval (2018)

[78] Gao, S., Sethi, A., Agarwal, S., Chung, T., Hakkani-Tur, D.: Dialog state tracking: A neural reading comprehension approach (2019)

[79] Gao, S., Sethi, A., Agarwal, S., Chung, T., Hakkani-Tür, D.Z.: Dialog state tracking: A neural reading comprehension approach. In: SIGdial (2019)

[80] Gatt, A., Krahmer, E.: Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. Journal of Artificial Intelligence Research **61**, 65–170 (2018)

[81] Gaulke, D., Kornbluh, D.: Interactive user interface to communication-enabled business process platforms method and apparatus (May 26 2015), uS Patent 9,043,407

[82] Gašić, M., Young, S.: Effective handling of dialogue state in the hidden information state pomdp-based dialogue manager. ACM Trans. Speech Lang. Process. **7**(3) (2011)

[83] Ghazvininejad, M., Brockett, C., Chang, M.W., Dolan, B., Gao, J., Yih, W.t., Galley, M.: A knowledge-grounded neural conversation model. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018)

[84] Goel, R., Paul, S., Hakkani-Tür, D.: Hyst: A hybrid approach for flexible and accurate dialogue state tracking. In: Interspeech (2019)

[85] Goel, R., Paul, S., Hakkani-Tür, D.: Hyst: A hybrid approach for flexible and accurate dialogue state tracking. In: Interspeech (2019)

[86] Goel, R., Paul, S., Hakkani-Tür, D.: Hyst: A hybrid approach for flexible and accurate dialogue state tracking. arXiv preprint arXiv:1907.00883 (2019)

[87] Gopalakrishnan, K., Hedayatnia, B., Chen, Q., Gottardi, A., Kwatra, S., Venkatesh, A., Gabriel, R., Hakkani-Tür, D., AI, A.A.: Topical-chat: Towards knowledge-grounded open-domain conversations. In: INTERSPEECH. pp. 1891–1895 (2019)

[88] Gordon-Hall, G., Gorinski, P.J., Lampouras, G., Iacobacci, I.: Show us the way: Learning to manage dialog from demonstrations. In: The Eight Dialog System Technology Challenge (DSTC-8) Workshop at AAAI (2020)

[89] Gozalo-Brizuela, R., Garrido-Merchan, E.C.: Chatgpt is not all you need. a state of the art review of large generative ai models. arXiv preprint arXiv:2301.04655 (2023)

[90] Gu, J.C., Ling, Z.H., Wu, Y., Liu, Q., Chen, Z., Zhu, X.: Detecting speaker personas from conversational texts. arXiv preprint arXiv:2109.01330 (2021)

[91] Guo, D., Tang, D., Duan, N., Zhou, M., Yin, J.: Dialog-to-action: Conversational question answering over a large-scale knowledge base. Advances in Neural Information Processing Systems **31** (2018)

[92] Guo, J., Shuang, K., Li, J., Wang, Z., Liu, Y.: Beyond the granularity: Multi-perspective dialogue collaborative selection for dialogue state tracking. arXiv preprint arXiv:2205.10059 (2022)

[93] Hakim, F.Z.M., Indrayani, L.M., Amalia, R.M.: A dialogic analysis of compliment strategies employed by replika chatbot. In: Third International Conference of Arts, Language and Culture (ICALC 2018). pp. 266–271. Atlantis Press (2019)

[94] Han, T., Huang, C., Peng, W.: Coreference augmentation for multi-domain task-oriented dialogue state tracking. arXiv preprint arXiv:2106.08723 (2021)

[95] Hashemi, H.B., Asiaee, A., Kraft, R.: Query intent detection using convolutional neural networks. In: International Conference on Web Search and Data Mining, Workshop on Query Understanding (2016)

[96] Heck, M., van Niekerk, C., Lubis, N., Geishauser, C., Lin, H.C., Moresi, M., Gašić, M.: Trippy: A triple copy strategy for value independent neural dialog state tracking. arXiv preprint arXiv:2005.02877 (2020)

[97] Heer, J., Agrawala, M.: Software design patterns for information visualization. IEEE transactions on visualization and computer graphics **12**(5), 853–860 (2006)

[98] Henderson, M., Thomson, B., Young, S.: Deep neural network approach for the dialog state tracking challenge. SIGDIAL- 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue, Proceedings of the Conference pp. 467–471 (2013)

[99] Henderson, M., Thomson, B., Young, S.: Word-based dialog state tracking with recurrent neural networks. In: Proceedings of the 15th annual meeting of the special interest group on discourse and dialogue (SIGDIAL). pp. 292–299 (2014)

[100] Henderson, M.S.: Discriminative methods for statistical spoken dialogue systems. Ph.D. thesis, University of Cambridge (2015)

[101] Henderson, M.S.: Discriminative methods for statistical spoken dialogue systems. Ph.D. thesis, University of Cambridge (2015)

[102] Hoegen, R., Aneja, D., McDuff, D., Czerwinski, M.: An end-to-end conversational style matching agent. In: Proceedings of the 19th ACM International Conference on Intelligent Virtual Agents. pp. 111–118 (2019)

[103] Holmes, S., Moorhead, A., Bond, R., Zheng, H., Coates, V., McTear, M.: Usability testing of a healthcare chatbot: Can we use conventional methods to assess conversational user interfaces? In: Proceedings of the 31st European Conference on Cognitive Ergonomics. pp. 207–214 (2019)

[104] Hu, B., Lu, Z., Li, H., Chen, Q.: Convolutional neural network architectures for matching natural language sentences. Advances in neural information processing systems **27** (2014)

[105] Hu, J., Yang, Y., Chen, C., He, L., Yu, Z.: Sas: Dialogue state tracking via slot attention and slot information sharing. In: Proceedings of the 58th annual meeting of the association for computational linguistics. pp. 6366–6375 (2020)

[106] Huang, P.S., He, X., Gao, J., Deng, L., Acero, A., Heck, L.: Learning deep structured semantic models for web search using clickthrough data. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 2333–2338 (2013)

[107] Hutchby, I., Wooffitt, R.: Conversation analysis. Polity (2008)

[108] Jain, M., Kota, R., Kumar, P., Patel, S.N.: Convey: Exploring the use of a context view for chatbots. In: Proceedings of the 2018 chi conference on human factors in computing systems. pp. 1–6 (2018)

[109] Jain, M., Kumar, P., Kota, R., Patel, S.N.: Evaluating and informing the design of chatbots. In: Proc. of the 2018 Designing Interactive Systems Conference (2018)

[110] John, R.J.L., Potti, N., Patel, J.M.: Ava: From data to insights through conversations. In: CIDR-8th Biennial Conference on Innovative Data Systems Research (2017)

[111] Joshi, C.K., Mi, F., Faltings, B.: Personalization in goal-oriented dialog. arXiv preprint arXiv:1706.07503 (2017)

[112] Jurafsky, D.: Speech & language processing. Pearson Education India (2000)

[113] Jurafsky, D., Martin, J.H.: Speech and Language Processing: Chatbots & Dialogue Systems. Third draft edn. (2020), https://web.stanford.edu/~jurafsky/slp3/

[114] Kalia, A.K., Telang, P.R., Xiao, J., Vukovic, M.: Quark: a methodology to transform people-driven processes to chatbot services. In: Service-Oriented Computing: 15th International Conference, ICSOC 2017, Malaga, Spain, November 13–16, 2017, Proceedings. pp. 53–61. Springer (2017)

[115] Kaneshige, T., Hong, D.: Predictions 2019: This is the year to invest in humans, as backlash against chatbots and ai begins. Forrester, November **8** (2018)

[116] Kar, R., Haldar, R.: Applying chatbots to the internet of things: Opportunities and architectural elements. arXiv preprint arXiv:1611.03799 (2016)

[117] Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., et al.: Chatgpt for good? on opportunities and challenges of large language models for education. Learning and Individual Differences **103**, 102274 (2023)

[118] Khayrallah, H., Sedoc, J.: Smrt chatbots: Improving non-task-oriented dialog with simulated multiple reference training. arXiv preprint arXiv:2011.00547 (2020)

[119] Khurana, D., Koli, A., Khatter, K., Singh, S.: Natural language processing: State of the art, current trends and challenges. Multimedia Tools and Applications pp. 1–32 (2022)

[120] Kim, S., Yang, S., Kim, G., Lee, S.W.: Efficient dialogue state tracking by selectively overwriting memory. arXiv preprint arXiv:1911.03906 (2019)

[121] Kirby, S., Tamariz, M., Cornish, H., Smith, K.: Compression and communication in the cultural evolution of linguistic structure. Cognition **141**, 87–102 (2015)

[122] Kobeissi, M., Assy, N., Gaaloul, W., Defude, B., Haidar, B.: An intent-based natural language interface for querying process execution data. In: 2021 3rd International Conference on Process Mining (ICPM). pp. 152–159. IEEE (2021)

[123] Koller, A., Baumann, T., Köhn, A.: Dialogos: Simple and extensible dialog modeling. Fachbereich Informatik (2018)

[124] Kottur, S., Moura, J.M., Parikh, D., Batra, D., Rohrbach, M.: Clevr-dialog: A diagnostic dataset for multi-round reasoning in visual dialog. arXiv preprint arXiv:1903.03166 (2019)

[125] Lee, S.: Structured discriminative model for dialog state tracking. In: Proceedings of the SIGDIAL Conference. pp. 442–451. Association for Computational Linguistics (2013)

[126] Lei, S., Liu, S., Sen, M., Jiang, H., Wang, X.: Zero-shot state tracking and user adoption tracking on schema-guided dialogue. In: Dialog System Technology Challenge Workshop at AAAI (2020)

[127] Lei, W., Jin, X., Kan, M.Y., Ren, Z., He, X., Yin, D.: Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1437–1447 (2018)

[128] Leo John, R.J., Patel, J.M., Alexander, A.L., Singh, V., Adluru, N.: A natural language interface for dissemination of reproducible biomedical data science. In: MICCAI-Medical Image Computing and Computer Assisted Intervention. pp. 197–205. Springer International Publishing (2018)

[129] Li, A.W., Jiang, V., Feng, S.Y., Sprague, J., Zhou, W., Hoey, J.: Aloha: Artificial learning of human attributes for dialogue agents. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 8155–8163 (2020)

[130] Li, J., Sun, A., Han, J., Li, C.: A survey on deep learning for named entity recognition. IEEE Transactions on Knowledge and Data Engineering **34**(1), 50–70 (2020)

[131] Li, L., Xu, C., Wu, W., Zhao, Y., Zhao, X., Tao, C.: Zero-resource knowledge-grounded dialogue generation. Advances in Neural Information Processing Systems **33**, 8475–8485 (2020)

[132] Li, Q., Li, P., Ren, Z., Ren, P., Chen, Z.: Knowledge bridging for empathetic dialogue generation. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 10993–11001 (2022)

[133] Li, S., Yan, H., Qiu, X.: Contrast and generation make bart a good dialogue emotion recognizer. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 11002–11010 (2022)

[134] Li, Y., Feng, A.X., Li, J., Mumick, S., Halevy, A., Li, V., Tan, W.C.: Subjective databases. arXiv preprint arXiv:1902.09661 (2019)

[135] Liao, L., Long, L.H., Ma, Y., Lei, W., Chua, T.S.: Dialogue State Tracking with Incremental Reasoning. TACL **9**, 557–569 (2021)

[136] Lin, W., Tseng, B., Byrne, B.: Knowledge-aware graph-enhanced GPT-2 for dialogue state tracking. CoRR (2021)

[137] Lipton, Z.C., Gao, J., Li, L., Li, X., Ahmed, F., Deng, L.: Efficient exploration for dialog policy learning with deep bbq networks\& replay buffer spiking. CoRR abs/1608.05081 (2016)

[138] Lison, P.: A hybrid approach to dialogue management based on probabilistic rules. Computer Speech and Language **34**(1), 232 – 255 (2015)

[139] Liu, H., Chen, M., Wu, Y., He, X., Zhou, B.: Conversational query rewriting with self-supervised learning. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 7628–7632. IEEE (2021)

[140] Liu, Q., Chen, Y., Chen, B., Lou, J.G., Chen, Z., Zhou, B., Zhang, D.: You impress me: Dialogue generation via mutual persona perception. arXiv preprint arXiv:2004.05388 (2020)

[141] López, A., Sànchez-Ferreres, J., Carmona, J., Padró, L.: From process models to chatbots. In: Advanced Information Systems Engineering: 31st International Conference, CAiSE 2019, Rome, Italy, June 3–7, 2019, Proceedings 31. pp. 383–398. Springer (2019)

[142] López, A., Sànchez-Ferreres, J., Carmona, J., Padró, L.: From process models to chatbots. In: Advanced Information Systems Engineering. Springer International Publishing (2019)

[143] Lowe, R., Pow, N., Serban, I., Charlin, L., Pineau, J.: Incorporating unstructured textual knowledge sources into neural dialogue systems. In: Neural information processing systems workshop on machine learning for spoken language understanding (2015)

[144] Lund, B.D., Wang, T.: Chatting about chatgpt: how may ai and gpt impact academia and libraries? Library Hi Tech News (2023)

[145] Luo, L., Huang, W., Zeng, Q., Nie, Z., Sun, X.: Learning personalized end-to-end goal-oriented dialog. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 6794–6801 (2019)

[146] Łupkowski, P., Ginz, J.: A corpus-based taxonomy of question responses. IWCS '13 (2013)

[147] Ma, Y., Nguyen, K.L., Xing, F.Z., Cambria, E.: A survey on empathetic dialogue systems. Information Fusion **64**, 50–70 (2020)

[148] Mahesh, B.: Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet] **9**, 381–386 (2020)

[149] Mairesse, F., Young, S.: Stochastic language generation in dialogue using factored language models. Computational Linguistics **40**(4), 763–799 (2014)

[150] Manning, C.D., Eric, M.: A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. In: EACL (2017)

[151] Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The stanford corenlp natural language processing toolkit. In: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations. pp. 55–60 (2014)

[152] Matthies, C., Dobrigkeit, F., Hesse, G.: An additional set of (automated) eyes: chatbots for agile retrospectives. In: 2019 IEEE/ACM 1st international workshop on bots in software engineering (BotSE). pp. 34–37. IEEE (2019)

[153] McLeod, S., Kruijff-Korbayova, I., Kiefer, B.: Multi-task learning of system dialogue act selection for supervised pretraining of goal-oriented dialogue policies. In: Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue. pp. 411–417. Association for Computational Linguistics (2019)

[154] McTear, M., Callejas, Z., Griol, D.: The Conversational Interface: Talking to Smart Devices. Springer Publishing Company, Incorporated, 1st edn. (2016)

[155] McTear, M.F.: Spoken dialogue technology: Enabling the conversational user interface. ACM Comput. Surv. **34**(1), 90–169 (2002)

[156] Mesnil, G., He, X., Deng, L., Bengio, Y.: Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In: Interspeech. pp. 3771–3775 (2013)

[157] Metallinou, A., Bohus, D., Williams, J.D.: Discriminative state tracking for spoken dialog systems. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 466–475 (2013)

[158] Mrkšić, N., Ó Séaghdha, D., Wen, T.H., Thomson, B., Young, S.: Neural belief tracker: Data-driven dialogue state tracking. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1777–1788. Association for Computational Linguistics (Jul 2017), https://www.aclweb.org/anthology/P17-1163

[159] Mrkšić, N., Séaghdha, D.O., Thomson, B., Gašić, M., Su, P.H., Vandyke, D., Wen, T.H., Young, S.: Multi-domain dialog state tracking using recurrent neural networks (2015)

[160] Mrkšić, N., Séaghdha, D.O., Wen, T.H., Thomson, B., Young, S.: Neural belief tracker: Data-driven dialogue state tracking. arXiv preprint arXiv:1606.03777 (2016)

[161] Murphy, G.C.: Beyond integrated development environments: adding context to software development. In: 2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER). pp. 73–76. IEEE (2019)

[162] Nickerson, R.S., Butler, S.F., Carlin, M.: Empathy and knowledge projection. The social neuroscience of empathy pp. 43–56 (2009)

[163] Ouyang, Y., Chen, M., Dai, X., Zhao, Y., Huang, S., Chen, J.: Dialogue state tracking with explicit slot connection modeling. In: Proceedings of the 58th annual meeting of the association for computational linguistics. pp. 34–40 (2020)

[164] Pei, J., Ren, P., de Rijke, M.: A cooperative memory network for personalized task-oriented dialogue systems with incomplete user profiles. In: Proceedings of the Web Conference 2021. pp. 1552–1561 (2021)

[165] Peng, B., Li, X., Li, L., Gao, J., Celikyilmaz, A., Lee, S., Wong, K.F.: Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 2231–2240. Association for Computational Linguistics (2017)

[166] Perez, J., Liu, F.: Dialog state tracking, a machine reading approach using memory network. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics. pp. 305–314. Association for Computational Linguistics (2017)

[167] Prabhavalkar, R., Rao, K., Sainath, T.N., Li, B., Johnson, L., Jaitly, N.: A comparison of sequence-to-sequence models for speech recognition. In: Interspeech. pp. 939–943 (2017)

[168] Ramírez, J., Baez, M., Berro, A., Benatallah, B., Casati, F.: Crowdsourcing syntactically diverse paraphrases with diversity-aware prompts and workflows. In: Advanced Information Systems Engineering: 34th International Conference,

CAiSE 2022, Leuven, Belgium, June 6–10, 2022, Proceedings. pp. 253–269. Springer (2022)

[169] Rasa: `https://rasa.com/` - Last accessed on 2022-12-15

[170] Rasa Technologies: The rasa core dialogue engine, `https://rasa.com/docs/rasa/core/about/`- Last accessed on 2020-03-15

[171] Rastogi, A., Gupta, R., Hakkani-Tur, D.: Multi-task learning for joint language understanding and dialogue state tracking. arXiv preprint arXiv:1811.05408 (2018)

[172] Rastogi, A., Hakkani-Tür, D., Heck, L.: Scalable multi-domain dialogue state tracking. In: 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU). pp. 561–568. IEEE (2017)

[173] Rastogi, A., Zang, X., Sunkara, S., Gupta, R., Khaitan, P.: Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset **34**(05), 8689–8696 (2020)

[174] Rastogi, A., Zang, X., Sunkara, S., Gupta, R., Khaitan, P.: Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 8689–8696 (2020)

[175] Rastogi, P., Gupta, A., Chen, T., Lambert, M.: Scaling multi-domain dialogue state tracking via query reformulation. In: NAACL. pp. 97–105 (2019)

[176] Raux, A., Eskenazi, M.: A finite-state turn-taking model for spoken dialog systems. NAACL '09 (2009)

[177] Ren, H., Xu, W., Zhang, Y., Yan, Y.: Dialog state tracking using conditional random fields. In: Proceedings of the SIGDIAL 2013 Conference. pp. 457–461 (2013)

[178] Ren, L., Ni, J., McAuley, J.: Scalable and accurate dialogue state tracking via hierarchical sequence generation. arXiv preprint arXiv:1909.00754 (2019)

[179] Ren, L., Ni, J., McAuley, J.: Scalable and accurate dialogue state tracking via hierarchical sequence generation. In: EMNLP (2019)

[180] Rizk, Y., Bhandwalder, A., Boag, S., Chakraborti, T., Isahagian, V., Khazaeni, Y., Pollock, F., Unuvar, M.: A unified conversational assistant framework for business process automation. arXiv preprint arXiv:2001.03543 (2020)

[181] Ruder, S., Vulić, I., Søgaard, A.: A survey of cross-lingual word embedding models. Journal of Artificial Intelligence Research **65**, 569–631 (2019)

[182] San Segundo, R., Fernández, F., Ferreiros, J., Lucas, J., Salazar, J.: Managing speaker identity and user profiles in a spoken dialogue system. Procesamiento del lenguaje natural (43), 77–84 (2009)

[183] Sankar, G.R., Greyling, J., Vogts, D., du Plessis, M.C.: Models towards a hybrid conversational agent for contact centres. In: Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology. p. 200–209 (2008)

[184] Santhanam, S., Shaikh, S.: A survey of natural language generation techniques with a focus on dialogue systems-past, present and future directions. arXiv preprint arXiv:1906.00500 (2019)

[185] Sarikaya, R., Hinton, G.E., Ramabhadran, B.: Deep belief nets for natural language call-routing. In: 2011 IEEE International conference on acoustics, speech and signal processing (ICASSP). pp. 5680–5683. IEEE (2011)

[186] Schaffer, S., Gustke, O., Oldemeier, J., Reithinger, N.: Towards chatbots in the museum. In: mobileCH@ Mobile HCI (2018)

[187] Sen, P.C., Hajra, M., Ghosh, M.: Supervised classification algorithms in machine learning: A survey and review. In: Emerging technology in modelling and graphics, pp. 99–111. Springer (2020)

[188] Shah, P., Hakkani-Tur, D., Heck, L.: Interactive reinforcement learning for task-oriented dialogue management. In: NIPS 2016 Deep Learning for Action and Interaction Workshop (2016)

[189] Shan, Y., Li, Z., Zhang, J., Meng, F., Feng, Y., Niu, C., Zhou, J.: A contextual hierarchical attention network with adaptive objective for dialogue state tracking. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 6322–6333 (2020)

[190] Shelar, H., Kaur, G., Heda, N., Agrawal, P.: Named entity recognition approaches and their comparison for custom ner model. Science & Technology Libraries **39**(3), 324–337 (2020)

[191] Shen, Y., He, X., Gao, J., Deng, L., Mesnil, G.: Learning semantic representations using convolutional neural networks for web search. In: Proceedings of the 23rd international conference on world wide web. pp. 373–374 (2014)

[192] Sheth, A., Yip, H.Y., Iyengar, A., Tepper, P.: Cognitive services and intelligent chatbots: current perspectives and special issue introduction. IEEE Internet Computing **23**(2), 6–12 (2019)

[193] Singh, J., Joesph, M.H., Jabbar, K.B.A.: Rule-based chabot for student enquiries. Journal of Physics: Conference Series **1228** (2019)

[194] Singh, S., Litman, D., Kearns, M., Walker, M.: Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. Journal of Artificial Intelligence Research **16**, 105–133 (2002)

[195] Smith, C., Crook, N., Boye, J., Charlton, D., Dobnik, S., Pizzi, D., Cavazza, M., Pulman, S., de la Camara, R.S., Turunen, M.: Interaction strategies for an affective conversational agent. In: Intelligent Virtual Agents (2010)

[196] Song, H., Wang, Y., Zhang, K., Zhang, W.N., Liu, T.: Bob: Bert over bert for training persona-based dialogue models from limited personalized data. arXiv preprint arXiv:2106.06169 (2021)

[197] Stent, A., Prasad, R., Walker, M.: Trainable sentence planning for complex information presentation in spoken dialog systems. In: Proceedings of the 42nd annual meeting on association for computational linguistics. p. 79. Association for Computational Linguistics (2004)

[198] Su, P., Gasic, M., Mrksic, N., Rojas-Barahona, L.M., Ultes, S., Vandyke, D., Wen, T., Young, S.J.: Continuously learning neural dialogue management. CoRR (2016)

[199] Sun, K., Zhu, S., Chen, L., Yao, S., Wu, X., Yu, K.: Hybrid dialogue state tracking for real world human-to-human dialogues. In: Interspeech. pp. 2060–2064 (2016)

[200] Tešanovic, A.: What is a pattern. Dr. ing. course DT8100 (prev. 78901/45942/DIF8901) Object-oriented Systems (2005)

[201] Thomson, B., Young, S.: Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. Computer Speech and Language **24**(4), 562 (Mar 2010)

[202] Tigunova, A.: Extracting personal information from conversations. In: Companion Proceedings of the Web Conference 2020. pp. 284–288 (2020)

[203] Tigunova, A., Yates, A., Mirza, P., Weikum, G.: Listening between the lines: Learning personal attributes from conversations. In: The World Wide Web Conference. pp. 1818–1828 (2019)

[204] Tseng, B.H., Bhargava, S., Lu, J., Moniz, J.R.A., Piraviperumal, D., Li, L., Yu, H.: Cread: Combined resolution of ellipses and anaphora in dialogues. arXiv preprint arXiv:2105.09914 (2021)

[205] Tu, T., Ping, Q., Thattai, G., Tur, G., Natarajan, P.: Learning better visual dialog agents with pretrained visual-linguistic representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5622–5631 (2021)

[206] Tur, G., Deng, L., Hakkani-Tür, D., He, X.: Towards deeper understanding: Deep convex networks for semantic utterance classification. In: 2012 IEEE international conference on acoustics, speech and signal processing (ICASSP). pp. 5045–5048. IEEE (2012)

[207] Vinyals, O., Le, Q.: A neural conversational model. arXiv preprint arXiv:1506.05869 (2015)

[208] Vodolán, M., Kadlec, R., Kleindienst, J.: Hybrid dialog state tracker (2015)

[209] Vodolán, M., Kadlec, R., Kleindienst, J.: Hybrid dialog state tracker with asr features (2017)

[210] Walker, M.A., Rambow, O.C., Rogati, M.: Training a sentence planner for spoken dialogue using boosting. Computer Speech & Language **16**(3-4), 409–433 (2002)

[211] Wallace, R.: The elements of aiml style. ALICE A. I. Foundation (2003)

[212] Wang, Y., Shen, Y., Jin, H.: A bi-model approach for handling unknown slot values in dialogue state tracking. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 8019–8023 (2020)

[213] Wang, Y., Huang, J.: Formal modeling and specification of design patterns using rtpa. International Journal of Cognitive Informatics and Natural Intelligence (IJCINI) **2**(1), 100–111 (2008)

[214] Wang, Z.: Extracting and Inferring Personal Attributes from Dialogue. University of Washington (2021)

[215] Wang, Z., Lemon, O.: A simple and generic belief tracking mechanism for the dialog state tracking challenge: On the believability of observed information. In: Proceedings of the SIGDIAL Conference. pp. 423–432. Association for Computational Linguistics, Metz, France (2013)

[216] Weizenbaum, J.: Eliza—a computer program for the study of natural language communication between man and machine. Communications of the ACM **9**(1), 36–45 (1966)

[217] Weizenbaum, J.: Eliza — a computer program for the study of natural language communication between man and machine. Commun. ACM **26**(1), 23–28 (1983)

[218] Wen, T.H., Gašić, M., Mrkšić, N., Su, P.H., Vandyke, D., Young, S.: Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 1711–1721. Association for Computational Linguistics (2015)

[219] Wen, T.H., Vandyke, D., Mrkšíc, N., Gašíc, M., Rojas-Barahona, L.M., Su, P.H., Ultes, S., Young, S.: A network-based end-to-end trainable task-oriented dialogue system. In: 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of Conference. vol. 1, pp. 438–449 (2017)

[220] Williams, J.: A critical analysis of two statistical spoken dialog systems in public use. In: Proceedings IEEE Workshop on Spoken Language Technology (SLT). IEEE Spoken Language Technology Workshop (2012)

[221] Williams, J.: Multi-domain learning and generalization in dialog state tracking. In: Proceedings of the SIGDIAL Conference. Association for Computational Linguistics (2013)

[222] Williams, J.D.: Incremental partition recombination for efficient tracking of multiple dialog states. In: 2010 IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 5382–5385 (2010)

[223] Williams, J.D., Asadi, K., Zweig, G.: Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL) (2017)

[224] Williams, J.D., Poupart, P., Young, S.: Factored partially observable markov decision processes for dialogue management pp. 76–82 (2005)

[225] Williams, J.D., Young, S.: Partially observable markov decision processes for spoken dialog systems. Computer Speech & Language **21**(2), 393 – 422 (2007)

[226] Wu, C.S., Madotto, A., Hosseini-Asl, E., Xiong, C., Socher, R., Fung, P.: Transferable multi-domain state generator for task-oriented dialogue systems. In: ACL (2019)

[227] Wu, C.S., Madotto, A., Hosseini-Asl, E., Xiong, C., Socher, R., Fung, P.: Transferable multi-domain state generator for task-oriented dialogue systems. arXiv preprint arXiv:1905.08743 (2019)

[228] Wu, C.S., Madotto, A., Lin, Z., Xu, P., Fung, P.: Getting to know you: User attribute extraction from dialogues. arXiv preprint arXiv:1908.04621 (2019)

[229] Wu, P., Zou, B., Jiang, R., Aw, A.: Gcdst: A graph-based and copy-augmented multi-domain dialogue state tracking. In: Findings of the Association for Computational Linguistics: EMNLP 2020. pp. 1063–1073 (2020)

[230] Wu, Y., Li, Z., Wu, W., Zhou, M.: Response selection with topic clues for retrieval-based chatbots. Neurocomputing **316**, 251–261 (2018)

[231] Xu, G., Lee, H., Koo, M., Seo, J.: Optimizing policy via deep reinforcement learning for dialogue management. In: 2018 IEEE International Conference on Big Data and Smart Computing (BigComp). pp. 582–589 (2018)

[232] Xu, P., Hu, Q.: An end-to-end approach for handling unknown slot values in dialogue state tracking. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1448–1457 (2018)

[233] Xu, P., Hu, Q.: An end-to-end approach for handling unknown slot values in dialogue state tracking. arXiv preprint arXiv:1805.01555 (2018)

[234] Yan, Z., Duan, N., Bao, J., Chen, P., Zhou, M., Li, Z., Zhou, J.: Docchat: An information retrieval approach for chatbot engines using unstructured documents. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 516–525 (2016)

[235] Yang, P., Huang, H., Mao, X.L.: Comprehensive study: How the context information of different granularity affects dialogue state tracking? arXiv preprint arXiv:2105.03571 (2021)

[236] Yang, X., Liu, J.: Dialog state tracking using long short-term memory neural networks. In: INTERSPEECH (2015)

[237] Yann, D., Tur, G., Hakkani-Tur, D., Heck, L.: Zero-shot learning and clustering for semantic utterance classification using deep learning. In: International Conference on Learning Representations (cited on page 28) (2014)

[238] Yannakakis, M.: Hierarchical state machines. In: TCS. pp. 315–330. Springer (2000)

[239] Yao, K., Peng, B., Zhang, Y., Yu, D., Zweig, G., Shi, Y.: Spoken language understanding using long short-term memory neural networks. In: 2014 IEEE Spoken Language Technology Workshop (SLT). pp. 189–194. IEEE (2014)

[240] Yao, K., Zweig, G., Hwang, M.Y., Shi, Y., Yu, D.: Recurrent neural networks for language understanding. In: Interspeech. pp. 2524–2528 (2013)

[241] Ye, F., Manotumruksa, J., Zhang, Q., Li, S., Yilmaz, E.: Slot self-attentive dialogue state tracking. In: Proceedings of the Web Conference 2021. pp. 1598–1608 (2021)

[242] Yoshino, T., Fukuchi, Y., Matsumori, S., Imai, M.: Chat, shift and perform: Bridging the gap between task-oriented and non-task-oriented dialog systems. arXiv preprint arXiv:2206.11813 (2022)

[243] Young, S., Gašić, M., Thomson, B., Williams, J.D.: Pomdp-based statistical spoken dialog systems: A review. Proceedings of the IEEE **101**(5), 1160–1179 (2013)

[244] Zamanirad, S.: Superimposition of natural language conversations over software enabled services. Ph.D. thesis, University of New South Wales, Sydney, Australia (2019)

[245] Zamanirad, S.: Superimposition of natural language conversations over software enabled services (2019)

[246] Zamanirad, S.: Superimposition of natural language conversations over software enabled services. Ph.D. thesis, University of New South Wales, Australia (2019)

[247] Zamanirad, S., Benatallah, B., Barukh, M.C., Casati, F., Rodriguez, C.: Programming bots by synthesizing natural language expressions into api invocations. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 832–837. IEEE (2017)

[248] Zamanirad, S., Benatallah, B., Rodriguez, C., Yaghoubzadehfard, M., Bouguelia, S., Brabra, H.: Hierarchical state machine based conversation model and services. Proc. CAiSE 2020

[249] Zhang, B., Xu, X., Li, X., Ye, Y., Chen, X., Wang, Z.: A memory network based end-to-end personalized task-oriented dialogue generation. Knowledge-Based Systems **207**, 106398 (2020)

[250] Zhang, J.G., Hashimoto, K., Wu, C.S., Wan, Y., Yu, P.S., Socher, R., Xiong, C.: Find or classify? dual strategy for slot-value predictions on multi-domain dialog state tracking (2019)

[251] Zhang, J.G., Hashimoto, K., Wu, C.S., Wan, Y., Yu, P.S., Socher, R., Xiong, C.: Find or classify? dual strategy for slot-value predictions on multi-domain dialog state tracking. arXiv preprint arXiv:1910.03544 (2019)

[252] Zhang, J.G., Hashimoto, K., Wu, C.S., Wan, Y., Yu, P.S., Socher, R., Xiong, C.: Find or classify? dual strategy for slot-value predictions on multi-domain dialog state tracking (2020)

[253] Zhang, Z., Huang, M., Zhao, Z., Ji, F., Chen, H., Zhu, X.: Memory-augmented dialogue management for task-oriented dialogue systems. ACM Transactions on Information Systems (TOIS) **37**(3), 1–30 (2019)

[254] Zhang, Z., Huang, M., Zhao, Z., Ji, F., Chen, H., Zhu, X.: Memory-augmented dialogue management for task-oriented dialogue systems. ACM Transactions on Information Systems (TOIS) (2019)

[255] Zhang, Z., Li, X., Gao, J., Chen, E.: Budgeted policy learning for task-oriented dialogue systems. arXiv preprint arXiv:1906.00499 (2019)

[256] Zhao, T., Eskenazi, M.: Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In: Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue. pp. 1–10. Association for Computational Linguistics (2016)

[257] Zhao, T., Lu, A., Lee, K., Eskenazi, M.: Generative encoder-decoder models for task-oriented spoken dialog systems with chatting capability. In: Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue. pp. 27–36. Association for Computational Linguistics (2017)

[258] Zhao, X., Wang, L., He, R., Yang, T., Chang, J., Wang, R.: Multiple knowledge syncretic transformer for natural dialogue generation. In: Proceedings of The Web Conference 2020. pp. 752–762 (2020)

[259] Zhou, K., Prabhumoye, S., Black, A.W.: A dataset for document grounded conversations. arXiv preprint arXiv:1809.07358 (2018)

[260] Zhu, S., Li, J., Chen, L., Yu, K.: Efficient context and schema fusion networks for multi-domain dialogue state tracking. In: Findings of the Association for Computational Linguistics. pp. 766–781. Association for Computational Linguistics, Online (2020)