Universidade de Aveiro
2020

André Cunha de Oliveira

Contributos para Suportar o Desenvolvimento de Sistemas de Diálogo

Contributions to Support the Development of Dialogue Systems

Universidade de Aveiro
2020

André Cunha de Oliveira

Contributos para Suportar o Desenvolvimento de Sistemas de Diálogo

Contributions to Support the Development of Dialogue Systems

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor António Teixeira, Professor associado c/ agregação do da Universidade de Aveiro, e do Doutor Samuel Silva, Professor Doutorado (Nível 2) do Instituto de Engenharia Electrónica e Telemática de Aveiro
.

o júri / the jury

presidente / president	Professor Doutor Joaquim Madeira Professor Auxiliar, Universidade de Aveiro
vogais / examiners committee	Professor Doutor Alberto Simões Professor Adjunto Convidado, Instituto Politécnico do Cávado e do Ave
	Professor Doutor António Joaquim da Silva Teixeira Professor Associado C/ Agregação, Universidade de Aveiro

agradecimentos / acknowledgements

Em primeiro lugar, gostaria de agradecer ao meu orientador Professor Doutor António Teixeira e ao meu coorientador Doutor Samuel Silva por toda a ajuda e todo o apoio que me deram ao longo desta dissertação.
Agradeço também aos colegas do laboratório do IEETA por terem participado nos testes do primeiro assistente.
Aos meus amigos de curso, agradeço por me terem acompanhado durante esta jornada, assim como aos meus amigos da Figueira.
Ao Diogo e ao Luís, agradeço por toda a disponibilidade e por me terem ajudado em todas as dificuldades.
Aos meus pais e ao meu irmão, quero deixar um especial agradecimento por todo o suporte e por terem tornado esta etapa possível.
Finalmente, quero agradecer à Maria, por tudo.

Resumo

Com o aumento da tecnologia presente na nossa rotina diária, uma área específica subiu exponencialmente. Os sistemas de diálogo são cada vez mais populares e úteis: proporcionam um acesso mais fácil e versátil a grandes e diversificados conjuntos de informação.
Hoje em dia existe uma vasta base de conhecimento sobre este tema, bem como diferentes sistemas capazes de executar inúmeras tarefas através do processamento de uma entrada de voz. Há também um conjunto crescente de ferramentas para o seu desenvolvimento.
Apesar dos avanços recentes, o desenvolvimento de sistemas de diálogo continua a ser um desafio. O principal objetivo desta tese é contribuir e tornar possível e simples o desenvolvimento de novos sistemas de diálogo que suportem português: selecionando, adaptando e combinando ferramentas existentes.
Baseado nas melhorias feitas nas ferramentas bases selecionadas, foram desenvolvidos dois sistemas de diálogo diferentes: o primeiro é um assistente dirigido a um ambiente Smart Home - uma das áreas que mais beneficiou com o desenvolvimento de sistemas de diálogo - e o segundo visando o turismo acessível.
O primeiro assistente foi desenvolvido alinhado com o projeto Smart Green Homes. Implicou a definição de cenários e requisitos que mais tarde ajudaram a definir a ontologia e o sistema. Outro requisito para este sistema foi a inclusão do sistema back-end desenvolvido anteriormente como parte do projeto Smart Green Homes.
O segundo assistente foi alinhado com o projeto ACTION na área do Turismo. Foi desenvolvido para utilizadores com necessidades de acessibilidade, por exemplo, deficiência motora ou invisualidade.

Abstract

With the increase of technology present in our daily routine, a specific area rose exponentially. Spoken dialogue systems are increasingly popular and useful: they provide an easier and more versatile access to large and diverse sets of information.
Nowadays there is a vast knowledge base regarding this topic, as well as different systems capable of performing numerous tasks just by simply processing voice input. There are also an increasing set of tools for their development. Despite recent advances, development of dialog systems continues to be challenging. The main objective of this thesis is to contribute and make possible and simple the development of new dialogue systems for Portuguese: by selecting, adapting and combining existing tools/frameworks.
Supported by the enhancements made to the selected basis framework, two different dialog systems were developed: the first is an assistant aimed at a Smart Home environment - one of areas that benefited the most with the development of dialog systems - and the second targeting accessible tourism.
The first assistant was developed aligned with the Smart Green Homes project. It implied the definition of scenarios and requirements that later helped defining the ontology and the system. Another requirement for this system was the inclusion of the back-end system developed previously as part of the Smart Green Homes project.
The second was aligned with the project ACTION in the area of Tourism. It was developed for users with accessibility needs, e.g., impaired movement or vision.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

Dialogue and communication are fundamental pieces that enable interaction with other people: it is something that is intrinsically associated with our life experience. During the course of recent years, there has been an evolution in how we use computer systems, and one of those ways is by using communication in the way we are most used to.

With the increase of pieces of technology present in our daily routine, a specific area rose exponentially. Dialogue systems are increasingly popular because of their usefulness: they provide an easier and more versatile access to large and diverse sets of information. But this information was already reachable, so what is so special about these systems?

These systems can do what no other system was able to do before: process what we tell them and take actions accordingly. This ability comes especially important when dealing with users with, for example, visual impairments which disables them from interacting with a system via a touch screen.

Nowadays there is a vast knowledge base regarding this topic, as well as different systems capable of performing numerous tasks just by simply processing voice input (Apple, 2011, Ferguson et al., 2010, Amazon, 2014). One of the areas that benefited the most with the development of these systems is smart home. This is an area that is intrinsically connected to another rising concept: the internet of things (IoT). One can consider an house to be smart when it has several devices that can be remotely accessed, monitored or controlled connected. These devices can go from simple smart plugs and bulbs to more sophisticated devices like an Amazon Alexa (Amazon, 2014). This device developed by Amazon allows several devices to be connected to a central system that can be accessed and controlled via speech.

Moreover, several companies use chatbots as a new interactive way to connect with users and have their needs met. The airline industry, for example, usually sees a lot of competition and is well known for poor customer experience, lack of communication and lot of passenger dissatisfaction. To elevate flying experience, one of the latest disruptors being embraced by

IT heads at large airline companies are AI-powered chatbots (Streebo, 2018). Users interact with chatbots as if they were talking with a human being, and can usually have their requests attended by these systems. Even though they might not be as accurate as if it was a human being addressing them, they are available anytime and are usually able to call the assistance of a human being whenever it can't process the request from the user.

Without realizing, many of us carry an assistant in our pockets everyday since every smartphone has them implemented in one way or another. Ultimately, systems of this nature can vary in how they are to be interacted with or in which platforms they are used in, but they have their core in common.

Due to the disruption of this area specifically during my academic career, combined with certain disciplines that aroused special interest (Artificial Intelligence, Multimodal Interaction, Machine Learning and Information Retrieval), the proposal of this dissertation was aligned both in my personal interests (interaction with computers) as well as in my academic interests.

1.2 Context

The presented work on this thesis is aligned with two ongoing projects, that provide the scenarios and requirements: the Smart Green Homes project and the ACTION project.

The Smart Green Home project [1] (SGH) is the product of an ongoing partnership between Universidade de Aveiro and Bosch Termotecnologia, S. A.. It aims at integrating different technologies and applications in a household (e.g., water heaters and lighting) providing the most comfort and well being to the user.

In the scope of this project, there was already developed a virtual assistant to address these objectives (Ketsmur et al., 2019). This assistant is able to extract intents from the Portuguese user input, execute user commands, and provide the user with information regarding house consumptions.

The project ACTION [2] (Acessible tourism: Co-creation of Tourism experIence thrOugh Web-based IntelligeNt System) is a project that intends to develop a Web-based Intelligent System to facilitate the co-creation of tourism experiences for an accessible tourism market, i.e., persons with disabilities, elderly and other people with functional limitations. It aims at responding to this challenge by developing an innovative instrument to provide communication, information sharing, suggestions based on users' profiles and knowledge. Ultimately, it encourages the development of accessible tourism while contributing to the promotion of more inclusive societies.

1.3 Objectives

The main objective of this thesis is to contribute and make possible and simple the development of new dialogue systems for Portuguese: by selecting, adapting and if needed

[1] `https://www.ua.pt/pt/smartgreenhomes/`
[2] `http://action.web.ua.pt/`

combining existing tools/frameworks. By exploring and testing different tools and resources, the limitations of each one can be pointed out, and, with that information it is possible to make the most accurate choice as to which should be used.

The second objective is to make proof-of-concepts in different domains, using the selected framework: one aimed at smart homes and other towards accessible tourism.

The smart home spoken dialogue system has as it main objective to give the user a way to communicate with its house: allowing the user to turn on and off devices that are connected to the system, see and change the temperature of the water and checking for consumptions regarding the three different resources water, gas and electricity. This system was developed in alignment with an existent project between Universidade de Aveiro and Bosch Thermotechnology Aveiro, the Smart Green Homes project, integrating some previous developed features.

In colaboration with Departmento de Turismo da Universidade de Aveiro, and also aligned with an existent project, the ACTION project, a second spoken dialogue system was also developed. As mentioned before, project ACTION has as its main objective to make tourism accessible for everyone. As so, this dialogue system was developed keeping in mind the most needy consumer: people that have any kind of need or disability. The goal of this system is to give the user information regarding accessible tourism, e.g., if a certain touristic point has accessibility for people with reduced mobility.

1.4 Challenges

When interacting with a system of any nature, a user always tries to do it the most natural way possible. As so, if there is the possibility of using the system in the user's mother tongue, the user will always tend to do it. When it comes to the development of spoken dialogue systems, this must always be something to consider by the developers. Every explored framework had limited support for developing systems for languages other than English.

So, the first challenge was to find suitable tools that allowed the development of systems that can interact with the user in Portuguese.

Another challenge found when testing different frameworks, was the low flexibility that most of them offered regarding dialogue management with small amount of data for machine learning.

Regarding the scope of accessible tourism, understanding the needs of impaired users and addressing to them while developing a proof-of-concept was also a challenge.

1.5 Structure

The remainder of this document presents all the work carried out to address the objectives mentioned above and is organized as follows:

Chapter 2 - Background and Related Work: This chapter addresses important concepts for the development of the thesis. It analyses the main aspects and characteristics of the communication and dialogues between humans, such as turn-taking and dialogue acts. It also describes the typical modules that compose a dialogue system, as well as different existent dialogue managers present in spoken dialogue systems. Finally, it presents the studied and explored frameworks, followed by a critical analysis and a selection of which were used.

Chapter 3 - Framework Emora+: Presents the adjustments and improvements made to the chosen frameworks, detailing and explaining the reasoning behind each alteration.

Chapter 4 - Spoken Dialogue System for a Smart Home: This chapter presents the first assistant developed, for a Smart Home.

Chapter 5 - Proof-of-Concept for Acessible Tourism: Consists on the presentation of the second proof-of-concept developed in the scope of this work, an assistant capable of providing information regarding accessible tourism.

Chapter 6 - Conclusions: Includes summary of the work, main results and suggestions for future work.

CHAPTER 2

Background and Related Work

This chapter addresses the background and the related work regarding dialogue systems and the spoken language itself. First, essential information on human communication is presented along with its properties. Following this initial introduction to the topic, we dismantle spoken dialogue systems explaining each component that make it up and how they allow interactions with humans, further followed by some system examples divided by their categories.

Finally, the studied tools that allow the development of these dialogue systems are explained in detail and are discussed to which of those were chosen and why.

2.1 Communication and Dialogue

Conversation between humans is a complex joint task that cannot be fully replicated with machines due to the limitations of the current technologies, making conversations between humans and machines simpler and more constrained than human conversations. However, it is crucial to understand human conversations and their characteristics in order to develop conversational agents to converse with humans.

There are certain properties that are common in every human to human conversation. Jurafsky and Martin (Jurafsky and Martin, 2009a) define that a conversation has five main properties:

- Turn-taking;
- Dialogue Acts;
- Language as Joint Action;
- Conversational Structure;
- Conversational Implicature.

The following subsections explain these properties with further detail.

2.1.1 Turn-Taking

Turn-taking is a characteristic that is associated with the dialogue itself: only one speaker must speak at a time and to do so it must possess the turn. Even though speakers can overlap each other while talking, the average total amount of overlaps is really small: studies show that it happens less than 5% (Levinson, 1983, cited by Jurafsky and Martin, 2009a).

The event of changing the speaker that possesses the turn is called turn-taking and is exemplified in table 2.1.

Table 2.1: Demonstration of turn-taking in a regular dialogue.

André:	How many new cases are there in Portugal today?	(turn 1)
Maria:	I don't know, let me check.	(turn 2)
Maria:	There are almost 4000 new cases today!	
André:	Things are only getting worse...	(turn 3)
Maria:	I know, when will this ever end...	(turn 4)

One must also consider some rules when orchestrating with the turn-taking on a dialogue. Studies from Sacks, Schegloff, and Jefferson (Sacks, Schegloff, and Jefferson, 1978, cited by Jurafsky and Martin, 2009a), defined the following turn-taking rules:

1. If during a turn the current speaker A selects speaker B as the next speaker then B must speak next;

2. If the current speaker doesn't select the next speaker, any other speaker may take the next turn;

3. If no speaker takes the next turn, the current speaker may take the next turn.

2.1.2 Dialogue Acts

In "How to Do Things With Words" (Austin et al., 1975), Austin et al. identify the existence of two types of sentences. The first one is constative: mainly used in sentences that can be identified as either true or false, i.e., "The food is good." or "That shirt is blue.". However, Austin et al. also identify a second type of sentence that can't be analyzed in the same way as the previous ones. Some examples, taken from the book, are the following:

- "I do (sc. take this woman to be my lawful wedded wife)" - as uttered in the course of the marriage ceremony;

- "I name this ship the Queen Elizabeth" - as uttered when smashing the bottle against the stem;

- "I give and bequeath my watch to my brother" - as occurring in a will;
- "I bet you sixpence it will rain tomorrow.".

These sentences do not describe events or things thus making no sense to be qualified as true or false. Austin et al. calls these types of sentences performative sentences. Verbs such as "name", that perform this type of action, are called performative verbs. Moreover, the author named these kinds of actions as "speech acts".

In the same book, Austin et al. divide the act of saying a sentence into three different acts: "to say something is to do something", "in saying something we do something", and "by saying something we do something". Hence, they claim that the utterance of any sentence in a real speech situation constitutes three kinds of acts:

- Locutionary act: the utterance of a sentence with a particular meaning;
- Illocutionary act: the act of asking, answering, promising, etc., in uttering a sentence;
- Perlocutionary act: the (often intentional) production of certain effects upon the feelings, thoughts, or actions of the addressee in uttering a sentence.

For example, the utterance "You can't do that." might have the illocutionary force of protesting and the perlocutionary effect of stopping the addressee from doing something.

Even though Austin et al. classified speech acts according to their illocutionary force, Searle's (Searle, 1975) approach in modifying the taxonomy created by Austin et al. became more known, and it suggests that speech acts can be classified into one of five major classes:

- Assertives: committing the speaker to something's being the case (suggesting, swearing, concluding);
- Directives: attempts by the speaker to get the addressee to do something (asking, ordering, requesting);
- Commissives: committing the speaker to some future course of action (promising, vowing, planning);
- Expressives: expressing the psychological state of the speaker about a state of affairs (thanking, apologizing, welcoming);
- Declarations: bringing about a different state of the world via the utterance ("I resign", "You're fired").

When considering a human-machine interaction using a dialogue system, any user interaction is characterized by an intention. In "CUED Standard Dialogue Acts" (Young, 2007), Young represents the user intention as a core set of primitive dialogue acts that fall into four different categories:

1. Inform: used by the speaker to convey one or more items of information;

2. Query or Request: which requests an answer to a specific question;
3. Confirmation: inviting explicitly or implicitly yes or no answers;
4. Housekeeping: used to maintain turn-taking such as hello or acknowledge acts.

In linguistic terms, a dialogue turn can be a composition of several primitive dialogue acts. In some spoken dialogue systems, however, dialogue turns are usually represented by one single act. This is compensated with the ability to allow a single dialogue act to contain multiple items. For example, the user utterance "I would like to eat some Portuguese food and listen to some Fado." would be understood by the system as

```
inform(food="Portuguese", music="Fado")
```

which is quite different to

```
inform(food="Portuguese"), inform(music="Fado").
```

The latter would indicate that the user intended to either inform that `food="Portuguese"` or that `music="Fado"` but not both.

Abbeduto, 1983, however, defined four major classes of dialogue acts:

- Constatives: answering, claiming, confirming, denying, disagreeing, or stating;
- Directives: advising, asking, forbidding, inviting, ordering, or requesting;
- Commissives: promising, planning, vowing, betting, or opposing;
- Acknowledgments: apologizing, greeting, thanking, or accepting and acknowledgment.

Taking this approach, a user asking a person or a dialogue system to do something is issuing a Directive act. Likewise, asking a question that requires an answer is also issuing a Directive. On the other hand, a user stating a constraint (like 'I need to travel in May') is issuing a Constative act. When the user thanks to the system he is issuing an Acknowledgment.

2.1.3 Language as Joint Action

One implication of joint action is that the speaker and the hearer must constantly establish common ground (Stalnaker, 2002, cited by Jurafsky and Martin, 2009a): the set of things that are mutually believed by both speakers, like ideas and facts. The need to achieve common ground means that the hearer must ground the speaker's utterances, i.e., the hearer understands the speaker's meaning and intention.

As pointed out by Clark, 1996, people need closure or grounding for actions that are not linguistic: an action is considered a joint action if two or more people coordinate to execute it successfully. Thus, dialogue can be classified as a joint action. Clark phrases this need for closure as follows, after Norman, 2002:

Principle of Closure: Agents performing an action require evidence, sufficient for current purposes, that they have succeeded in performing it.

To achieve closure, Clark and Schaefer (Clark and Schaefer, 1989, cited by Jurafsky and Martin, 2009a) define that each joint action is divided into two phases: the presentation and the acceptance. In the first phase, a speaker presents the hearer with an utterance, performing a sort of speech act. In the acceptance phase, the hearer has to ground the utterance, indicating to the speaker if he understood.

In the same article, Clark and Schaefer discuss five main types of methods that a hearer B can use to ground the speaker A's utterance. Listed from weakest to strongest, the methods are:

1. Continued attention: B shows she is continuing to attend and therefore remains satisfied with A's presentation;

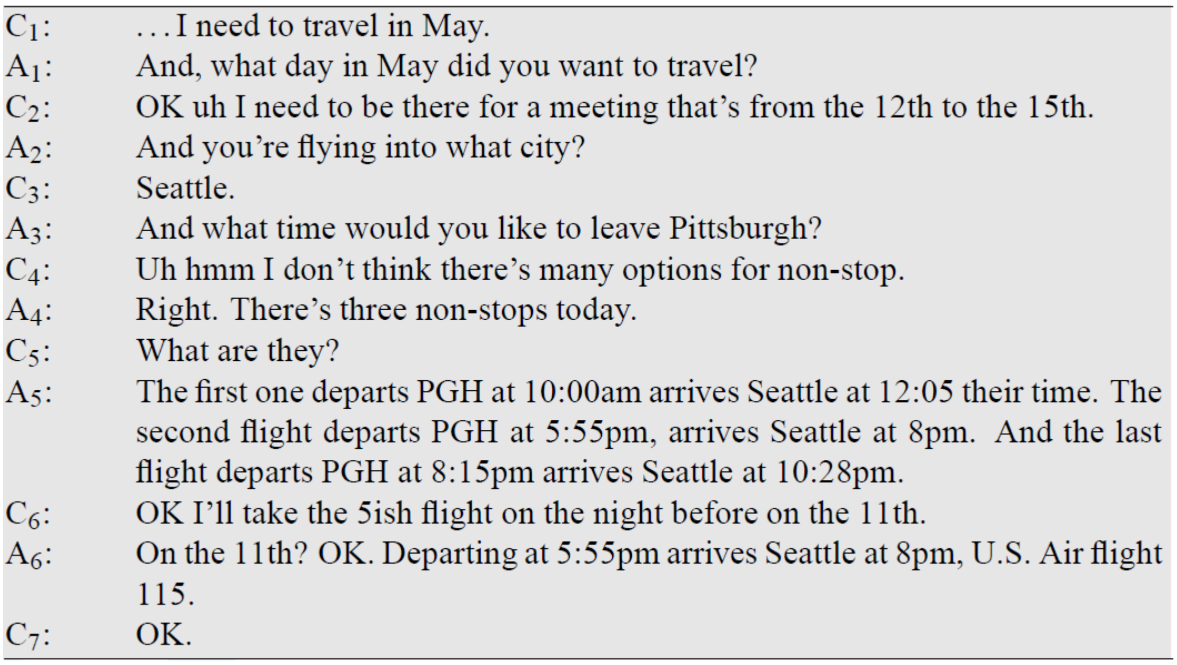
2. Next contribution: B starts in on the next relevant contribution;

3. Acknowledgement: B nods or says a continuer like "uh-huh", "yeah", etc;

4. Demonstration: B demonstrates all or part of what she has understood A to mean, for example, by reformulating A's utterance;

5. Display: B displays verbatim all or part of A's presentation.

Examples of grounding between a human travel agent and a human client can be found in figure 2.1.



C_1:	. . . I need to travel in May.
A_1:	And, what day in May did you want to travel?
C_2:	OK uh I need to be there for a meeting that's from the 12th to the 15th.
A_2:	And you're flying into what city?
C_3:	Seattle.
A_3:	And what time would you like to leave Pittsburgh?
C_4:	Uh hmm I don't think there's many options for non-stop.
A_4:	Right. There's three non-stops today.
C_5:	What are they?
A_5:	The first one departs PGH at 10:00am arrives Seattle at 12:05 their time. The second flight departs PGH at 5:55pm, arrives Seattle at 8pm. And the last flight departs PGH at 8:15pm arrives Seattle at 10:28pm.
C_6:	OK I'll take the 5ish flight on the night before on the 11th.
A_6:	On the 11th? OK. Departing at 5:55pm arrives Seattle at 8pm, U.S. Air flight 115.
C_7:	OK.

Figure 2.1: Part of a conversation between a travel agent (A) and client (C), taken from Jurafsky and Martin, 2009a.

Utterance A_1, in which the agent repeats *in May*, shows the strongest from of grounding, in which the hearer displays their understanding by repeating the verbatim part of the speakers words (Figure 2.2):

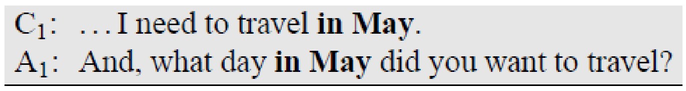

C_1: ... I need to travel **in May**.
A_1: And, what day **in May** did you want to travel?

Figure 2.2: Specific grounding example, taken from Jurafsky and Martin, 2009a.

2.1.4 Conversational Structure

The area known as Conversational Analysis has as its main goal to study how interactions occur during a conversation as well as how they are structured. This area was initially studied by Sacks, Schegloff and Jefferson in the 60's (Goodwin and Heritage, 1990, cited by Jurafsky and Martin, 2009a), and indicated that the utterances between participants in a conversation tend to appear in pairs during the dialogue. These pairs were later called adjacency pairs (Hutchby and Wooffitt, 1998).

Schegloff and Sacks (Schegloff and Sacks, 1973) established a rule for the adjacency pairs: after the speaker produces the first production of the pair, the hearer must produce the second element of the pair accordingly to the first.

These pairs are easy to identify in the course of a dialogue. For example, when someone asks something the next contribution to the conversation is usually the hearer's answer. Likewise, a proposal is usually followed by an acceptance or refusal, a wish is followed by a grant, etc.

Still in the topic, there is a preference between pairs that can be used in the same dialogue. Taking an invitation as an example, both an acceptance and a refusal are valid but there is a preference for acceptance. Furthermore, the acceptance tends to be more natural and faster while the refusal tends to be longer, have hesitations, and require explanations (Pomerantz, 1985).

2.1.5 Conversational Implicature

Conversational Implicature was a term created by Grice (Grice, 1975) to represent a subclass of unconventional inferences used in dialogue and to explain them, Grice suggests that conversations follow the cooperative principle.

A dialogue, as explained in the sections above, is a joint action that has a common direction and purpose. However, in the course of the dialogue, changes in directions can occur. Some of these changes of directions can be considered inadequate. To prevent this, Grice formulated his known cooperative principle that states: "Make your contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which you are engaged".

Along with this principle, he proposed the following four maxims:

- Maxim of Quantity: Be exactly as informative as is required:
  1. Make your contribution as informative as is required (for the current purposes of the exchange).
  2. Do not make your contribution more informative than is required.
- Maxim of Quality: Try to make your contribution one that is true:
  1. Do not say what you believe to be false.
  2. Do not say that for which you lack adequate evidence.
- Maxim of Relevance: Be relevant.
- Maxim of Manner: Be perspicuous:
  1. Avoid obscurity of expression.
  2. Avoid ambiguity.
  3. Be brief (avoid unnecessary prolixity).
  4. Be orderly.

2.2 Dialogue Systems

A dialogue system (Arora, Batra, and Singh, 2013) is a computer program that is capable of communicating with a human user naturally. These systems provide an interface between the user and the computer-based application and can be used in several different devices such as telephones, cars, and web browsers using different types of interfaces: be it CUI (Character User Interface), GUI (Graphical User Interface), VUI (Voice User Interface), multi-model, etc.

The way the user interacts with the system depends on the context of the system itself: while some justify the usage of text, others rely on the interaction using the voice. Note that some systems are also developed to be multimodal, i.e., systems that process two or more combined user input and/or output modes such as speech, touch, gestures, body movements, etc.

Dialogue systems may have different architectures but they all have the same set of components (Jurafsky and Martin, 2009a, Bohus and Rudnicky, 2009, Arora, Batra, and Singh, 2013, Ketsmur et al., 2018):

1. Input (Speech) Recognition;
2. Natural Language Understanding;
3. Dialogue Management;
4. Response (Language) Generation;
5. Output Renderer (Speech Synthesis).

Each of the enumerated components above has its own module to operate and how they interoperate is illustrated in Figure 2.3.

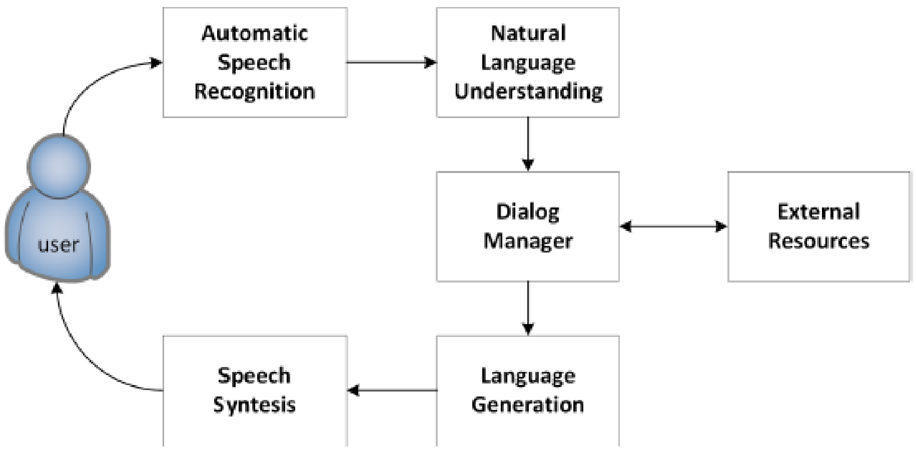


Figure 2.3: General architecture of a spoken dialogue system (Ketsmur et al., 2018).

2.2.1 Input Recognition

When considering the interaction between a human and a machine the first task we need to consider is the recognition of the user input. The system can understand the user input using a module called Automatic Speech Recognition that receives the human voice as an input and translates them into a sequence of words. The difficulty of this task depends on the vocabulary it was designed to understand.

A good ASR module stands out for its speed, allowing real-time answers, for the vocabulary they can understand, and for the way they can bypass any background noise.

An Automatic Speech Recognition module is only used in systems that are developed to recognize speech inputs, and so, can be discarded in systems that, for example, are intended to understand text inputs. However, there are other types of Input Recognition modules, e.g, recognizing writing using a smart pen on touch screen devices, or gesture recognizing.

The main goal of this component is to convert the input into a sequence of words, i.e., a text form of the initial input.

2.2.2 Natural Language Understanding

Following the previous task, the system has to be capable of understanding the user input, i.e., extracting the semantics behind the sequence of words generated by the Automatic Speech Recognition module. There are several approaches one can take when developing an Natural Language Understanding (NLU) module and it always depends on the system's scope.

One common approach is known as frame-based and works similarly to filling the blank spaces of a form: the system tries to extract from the sequence of words the information necessary to fill those blank spaces, allowing the system to further access a database. Thus a sentence like "Show me the afternoon flights from New York and San Francisco on Friday" would correspond to the filled-out frame presented in the table 2.2.

Another approach that can be found in several systems splits this task into two separate parts: first and foremost they perform a syntax analysis of the sequence of words and then

Table 2.2: Form for a Air Travel System.

Camp	Information
Origin	New York
Destiny	San Francisco
Date	Friday
Time	Afternoon

try to extract the semantics using pre-built grammars (Issar and Ward, 1993, Ward and Issar, 1994, cited by Jurafsky and Martin, 2009a).

2.2.3 Dialogue Management

The dialogue manager is the module capable of producing the best possible answer to the semantics extracted by the NLU, always keeping in mind the structure of the conversation, as well as other information regarding it, such as the user and the context of the conversation.

Nowadays there are several approaches the dialogue management. Some systems are based on finite state machines, others use a frame-based approach. Another approach worth mentioning is the information state-based systems: in these systems the main focus of the dialogue manager is to work around the context of the dialogue knowing when to update it or to decide which action should be taken at each moment. These systems are further detailed in section 2.3.5.

In January 1996, Weizenbaum (Weizenbaum, 1966) released one of the first dialogue systems that would later become really popular named ELIZA. ELIZA didn't maintain a context of dialogue since it mainly searched for keywords on the user input and applied transformations on the text it received. ELIZA had the role of a psychoanalyst and although the limitations it had, it was capable of generating dialogues like shown in the table 2.3.

Table 2.3: Example of Dialogue Generated by ELIZA.

USER:	Men are all alike.
ELIZA:	In what way?
USER:	They're always bugging us about something or other.
ELIZA:	Can you think of a specific example?
USER:	Well my boyfriend made me come here.
ELIZA:	Your boyfriend made you come here.
USER:	He says I'm depressed much of the time.
ELIZA:	I am sorry to hear that you are depressed.
	...
ELIZA:	Who else in your family takes care of you?
USER:	My father.
ELIZA:	Your father.
USER:	You are like my father in some ways.

2.2.4 Response Generation

The natural language generation is divided into two main phases: first, it decides on what to say, that in most systems is decided by the dialogue manager, and then how to say it, the language generation.

The simplest way to generate the language is to take the template approach. This approach is based on having pre-established sentences containing variables that are filled by the generator. Taking the example presented in section 2.2.2, the following sentences:

What time do you want to leave `CITY-ORIGIN`?

Will you return to `CITY-ORIGIN` from `CITY-DESTINY`?

could be follow-up confirmation questions in which the values of `CITY-ORIGIN` and `CITY-DESTINY` would be inserted into the template (pre-established) sentence.

Another way is to have the dialogue manager generate the semantics, giving the Natural Language Generation (NLG) the task of transforming the semantics into natural language. This approach is the one used in Plato (Papangelis et al., 2020) (see section 2.4.7).

2.2.5 Output Renderer

The last module of the system has the task of taking the text produced by the NLG and producing speech. It uses a technology named Text-To-Speech (TTS) that has a complex task and so usually divided it into sub-tasks.

Firstly the TTS has to perform a phonetic transcription of the text to be read. Then, all the rhythm and tones have to be taken into consideration. Lastly, the Speech Synthesis sub-module transforms the symbolic information generated in the steps mentioned before into speech (Dutoit, 1997).

2.3 Types of Dialogue Management

Spoken Dialogue Systems are increasingly being used as a result of the development of new, interactive, and multi-modal ways to use systems that surround us. These may come in several forms, from small chatbots to big and widely known conversational assistants like Apple's Siri (Apple, 2011), Microsoft's Cortana (Microsoft, 2014), or even Amazon's Alexa (Amazon, 2014).

Besides the main purpose of each system, they also can differ on how they are built on their core, specifically regarding the Dialogue Manager. As mentioned above, the Dialogue Manager is what makes the system capable of maintaining a conversation with the user rather than just being able to have a one-turn interaction with him. This is mainly true when the Dialogue Manager has the ability to maintain contexts when interacting with the user, i.e., remembering previous conversations and provide answers accordingly.

Regarding the internal architecture of dialogue systems, there isn't a consensus on how to divide them. McTear (McTear, 2004) divides them in: finite-state (or graph) based systems, frame based systems, and agent based systems. Daniel Jurafsky and James H. Martin (Jurafsky and Martin, 2009b), however, consider three different types:

1. Finite State Systems;

2. Frame-Based Systems;

3. Other Advanced Systems such as:

    - Systems based on Belief, Desire and Intention Models (BDI) that are used in dialogue systems based on plans;

    - Information State systems that are capable of maintaining information related to the context of the conversation and take decisions according to that context;

    - Markov Decision Process (MDP) systems that define the behavior of the Dialogue Manager through the usage of probabilities.

The latter is the division that will be taken into account on the following sections.

2.3.1 Finite State Systems

In a finite state based system, there is a state machine coded onto the system as well as every possible state transition. The transition from one state to another happens when the user provides input. One simple interaction with a basic finite-state based system is presented in the table 2.4.

Table 2.4: Example of a Basic Finite-State Based System, taken from (McTear, 2004)

SYSTEM:	What is your destination?
USER:	London.
SYSTEM:	Was that London?
USER:	Yes.
SYSTEM:	What day do you want to travel?
USER:	Friday.
SYSTEM:	Was that Sunday?
USER:	No.
SYSTEM:	What day do you want to travel?

The main advantage of this approach is that the system knows to which question the user is answering at every interaction, allowing a better and more suited interaction with the system.

One issue that can arise when using this type of system is that the number of possible transitions on each state is equal to every possible user answer, which can lead to an exponential rise in the number of states. Furthermore, this issue can lead to very long dialogues due to how these systems verify the information provided by the user. To prevent this, these systems usually restrict the user's input to predetermined words and phrases.

These systems can, however, accept more natural inputs when paired with natural language components. One example of a more suited but flawed interaction is presented in table 2.5, which represents a demo version of the *Nuance Automatic Banking* system (McTear, 2004), and has the dialogue flow represented in figure 2.4.

In the mentioned dialogue, and more specifically in $User_2$, the user is questioned about the payment amount, to which he responds the payment amount and the payment date. However, the system was only looking for the amount of the payment, and that is why it requests the payment date on the next interaction, $System_3$. Moreover, the extra information the user gave the system in $User_2$, made the system misunderstand the value that the user intended to pay, as seen in $System_4$: the system recognized "next Monday" not as a date but as part of the amount, i.e., as "and sixty-one pence". As shown in $System_5$-$User_7$, the system is able to recover from the error by getting the user to reconfirm the payment amount.

Systems using this approach of dialogue management are usually designed to have the initiative in the dialogue, which is exemplified in tables 2.4 and 2.5: the system is constantly the one who is making the questions.

Table 2.5: Example of a Finite-State Based System - Nuance Automatic Banking System (McTear, 2004)

$System_0$:	What would you like to do?
$User_0$:	Pay a bill.
$System_1$:	What company would you like to pay?
$User_1$:	Abbey National.
$System_2$:	How much would you like to pay?
$User_2$:	One hundred pounds next Monday.
$System_3$:	What date would you like the payment to be made on?
$User_3$:	Next Monday.
$System_4$:	You want to pay Abbey National one hundred pounds and sixty-one pence on April 5th? Is this correct?
$User_4$:	No.
$System_5$:	Would you like to change the company, amount, or date?
$User_5$:	Amount.
$System_6$:	How much would you like to pay?
$User_6$:	One hundred pounds.
$System_7$:	You want to pay Abbey National one hundred pounds on April 5th? Is this correct?
$User_7$:	Yes.

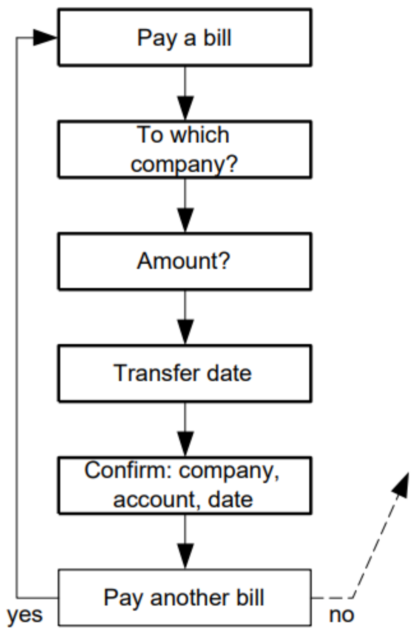


Figure 2.4: Dialogue flow for payment of a bill, from (McTear, 2004).

2.3.2 Frame-Based Systems

In a frame-based system (also known as the slot-filling system), the system tries to fill different slots in a template by making questions to the user in order to perform a task, usually by interacting with a database. In this type of system, the flow of the dialogue is not predetermined as it depends on the content of the user's input and which slots the system has been capable to fill. For example:

Table 2.6: Example of a Basic Frame-Based System, taken from (McTear, 2004)

SYSTEM:	What is your destination?
USER:	London.
SYSTEM:	What day do you want to travel?
USER:	Friday.
SYSTEM:	What is your destination?
USER:	London on Friday around 10 in the morning.
SYSTEM:	I have the following connection...

Initially the user provides one item of information at a time and the system performs similarly to a state-based system. However, when the user provides more than the requested information, the system accepts it and checks if any additional item of information is required. In the example, since it has all the required information, the system starts searching the database for a connection.

Frame-based systems are capable of producing more fluent dialogues when comparing to finite-state based systems. Table 2.7 shows a dialogue from the *Philips Automatic Train Timetable Information* system (Aust et al., 1995).

In the presented example, the first system interaction ($System_0$) ends with an open question, "How can I help you?", which allows the user to answer naturally. In $System_2$, the system applies implicit verification to make sure it has filled the required slots correctly, and the user corrects it as it recognized wrong the time to travel. After correctly filling every slot, the system is capable of accessing the database and the dialogue flows onwards.

Table 2.7: Example of a Frame-Based System - Philips Automatic Train Timetable Information System, from (Aust et al., 1995)

$System_0$:	Good morning. This is the automatic timetable information service. How can I help you?
$User_0$:	I need a connection from Hamburg to Munich.
$System_1$:	When would you like to go from Hamburg to Munich?
$User_1$:	Next Friday at eight o'clock.
$System_2$:	So you want to travel on Sunday the 20th of November at eight o'clock a.m.?
$User_2$:	No, on Friday.
$System_3$:	So you want to travel on Friday the 18th of November at eight o'clock a.m.?
$User_3$:	Yes.
$System_4$:	There is the following connection. With IntercityExpress 793, departure from Hamburg at 8:02, arrival in Munich at 14:05. Do you want me to repeat the connection?
$User_4$:	No, but I need a train for the return trip on Sunday.
$System_5$:	At what time would you like to go from Munich to Hamburg on Sunday the 20th of November?
$User_5$:	I have to arrive in Hamburg late in the evening.
$System_6$:	So you want to arrive in Hamburg between nine p.m. and eleven p.m.?
$User_6$:	Yes.
$System_7$:	There are three connections. This is the first one: With IntercityExpress 880, departure from Munich at 15:54, arrival in Hamburg at 21:58. Do you want the next connection?
$User_7$:	No, thanks, that one is fine.
$System_7$:	Do you need yet another connection?
$User_7$:	No, thank you.
$System_7$:	We wish you a pleasant journey. Thank you for calling. Good bye.

2.3.3 Belief, Desire and Intention based Systems

Systems based on Belief, Desire and Intention models (BDI) use planning algorithms to better understand and manage the dialogue (Bratman, Israel, and Pollack, 1988). Systems based on these models try to work on three different levels:

- Try to infer the user's objective;

- Try to anticipate any obstacle that can arise;

- Try to give answers in order to achieve that objective.

The original BDI model is specialized in conversational actions, as illustrated in figure 2.5 (Allen, 1995). Based on its current beliefs about the domain, including nested beliefs about shared knowledge, and the discourage obligations that each conversational agent has, the agent selects communicative goals, decides what speech act to perform next, generates an utterance, analyses the manager's response, and updates its beliefs about the discourse state and its own discourse obligations.

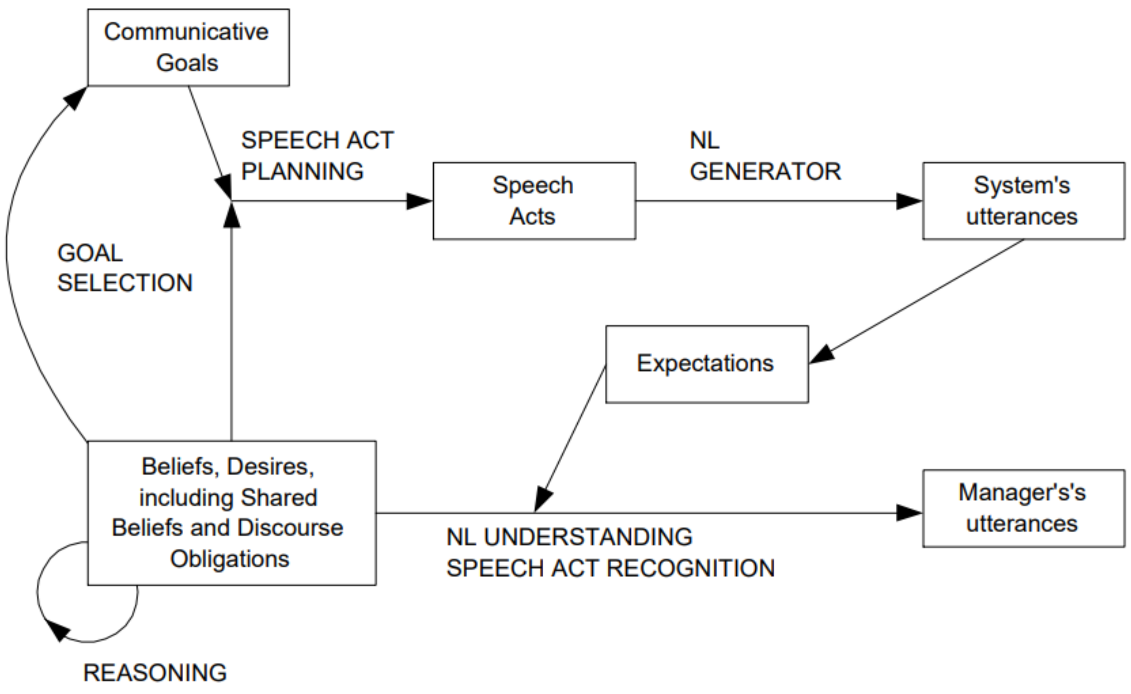


Figure 2.5: The BDI model of conversational agency, taken from (McTear, 2004).

However, some issues have to be dealt with when using these systems. To infer the user's objective, the system has to be capable of recognizing the dialogue acts in each sentence. If the system recognizes wrong dialogue acts, then it will infer the wrong user's objective and has to use extra resources to recover from it.

Discourse obligations are an important element of the conversational agent (Traum and Allen, 1994). They argue that models that are based on the analysis of the user's intention and the construction of a cooperative plan between agents do not explain why the agents have to answer when there isn't a cooperative plan or when they don't know the answer. Traum and Allen explain this behavior by standing out the difference between the agent's intentions and its obligations. Table 2.8 presents some discourse obligation rules.

Taking the example of a request by S_1, there is a discourse obligation on S_2 to respond to the request, even though not necessarily to accept it.

Table 2.8: Example of Discourse Obligations, taken from (Traum and Allen, 1994)

Source of Obligation	Obliged Action
S_1 Accept or Promise A	S_1 achieve A
S_1 Request A	S_2 address Request: accept A or reject A
S_1 YesNo Question whether P	S_2 Answer-if P
S_1 WH-Question $P(x)$	S_2 Inform-ref x
Utterance not understood or incorrect	Repair utterance

2.3.4 Markovian Decision Process based Systems

A Markovian Decision Process (MDP) is characterized by (Jurafsky and Martin, 2009a):

- A set of states S in which the agent can be in at a given moment;
- A set of actions A that the agent can take;
- A rewarding function $r(a, s)$ that the agent receives for taking an action in a certain state.

Given the previously mentioned factors, a policy π can be computed, which specifies which action $a \in A$ the agent should take when in a given state $s \in S$, so it can receive the best reward.

There are several ways to define the best policy to use, the most common being using the recursive *Bellman* equation.

What the *Bellman* equation says is that the expected cumulative reward for a given state/action pair is the immediate reward for the current state plus the expected discounted utility of all possible next states: this discount is used so that the agent values more the current rewards.

To use the *Bellman* equation, it is needed a model that defines:

- The probability that a state/action pair (s, a) will lead to a new state s';
- A good estimate of its reward $R(s, a)$.

To determine both of the referred parameters it is needed enough labeled data, and there are two methods to obtain it (Jurafsky and Martin, 2009a):

1. The first approach, used by (Singh et al., 2002) to optimize the dialogue policy adopted in the *NJFun* system, an experimental spoken dialogue system that provides users with access to information about fun things to do in New Jersey. This method is based on adjusting the number of states and policies to the minimum and creating a dialogue system that explores state space by generating random conversations. Probabilities can then be set from this corpus of conversations.

2. The second method is to build a simulated user that interacts with the system millions of times, allowing the system to learn the state transition and reward probabilities from this corpus. This is the approach used by (Levin, Pieraccini, and Eckert, 2000) in their MDP model with reinforcement learning system, an air travel information system. The simulated user was used to interacted with the system for tens of thousands of conversations, leading to an optimal dialogue policy.

This is the approach used in PyDial (Ultes et al., 2017), explained in section 2.4.4.

2.3.5 Information-State based Systems

In Information-State based systems, the context (information-state) of the dialogue is essential to the dialogue itself. The information-state keeps the progress of the main properties of a dialogue at each given moment, which facilitates the dialogue manager to make the best decisions.

An Information-State based system must contain the following elements (Larsson and Traum, 2000) (Traum and Larsson, 2003):

- A description of the informational components of the dialogue such as participants, common ground, linguistic and intentional structure, etc.;
- A formal representation of the above components;
- A set of dialogue moves that can trigger the update of the information state;
- A set of update rules that define when and how to update the information state;
- An update strategy for deciding which rules to select at a given point, from the set of applicable ones. This strategy can be as simple as "pick the first rule that applies" to more sophisticated arbitration mechanisms.

This type of dialogue system is the base of TrindiKit (Larsson and Traum, 2000) explained in section 2.4.1 and is also one way to utilize Emora (Finch and Choi, 2020) that is explained in section 2.4.6.

2.3.6 Multiple Strategy Systems

There are systems capable of applying multiple strategies. In these systems, the dialogue manager has the tools to change the strategy according to various circumstances. The main purpose of these systems is to take what is best of every single strategy previously mentioned, making the human-machine interaction as natural as possible (Chu et al., 2005).

2.4 Tools for Dialogue Systems Development - Representative Examples of the State-of-the-Art

The scope of this dissertation implied a vast search of tools and frameworks that allow the development of dialogue systems, and this section presents information regarding the most relevant to the objectives of the presented work. Due to their relevance today, they constitute a good sample of the state of the art.

This section presents the tools that were studied in the scope of this project: TrindiKit, OpenDial, Rasa, PyDial, ICECAPS, Emora and Plato; presented in chronological order, regarding their release date.

2.4.1 TrindiKit

TrindiKit (Larsson and Traum, 2000) is a toolkit for building and developing dialogue managers based on information state, released in 2000. As explained in section 2.3.5, one of the main functions that the developer has to address when developing such systems is how to update the information state. TrindiKit does so through *dialogue moves*, classifying this toolkit as a *Dialogue Move Engine* (DME) since its main functions are updating information state based on the observance of moves and selecting moves to be performed.

A complete dialogue system using TrindiKit would need the following modules:

- User interface: to receive input from and present output to the user;
- Interpretation: to calculate from the user input which dialogue moves have been performed, adding them to a part of the information state;
- Generation: to take the contents of the information state and produce the output;
- Control: to wire together all the other modules.

Figure 2.6 shows the architecture of TrindiKit, on which is clear to see the DME module, the IS (information state) module as well as every module that a dialogue system needs to implement.

To ease the process of creating a new dialogue system, TrindiKit can provide the developer with standard modules for the input, interpretation, generation, and output modules depicted in the architecture.

Being the oldest tool presented in this section, TrindiKit was implemented in a non-free dialect of the programming language, Prolog. To keep TrindiKit up-to-date with the more common programming languages, it was adapted into an open-source Python package by Ljunglöf, who tried to remain TrindiKit close to the original formulation, while making the most of Python classes and objects (Ljunglöf, 2009).

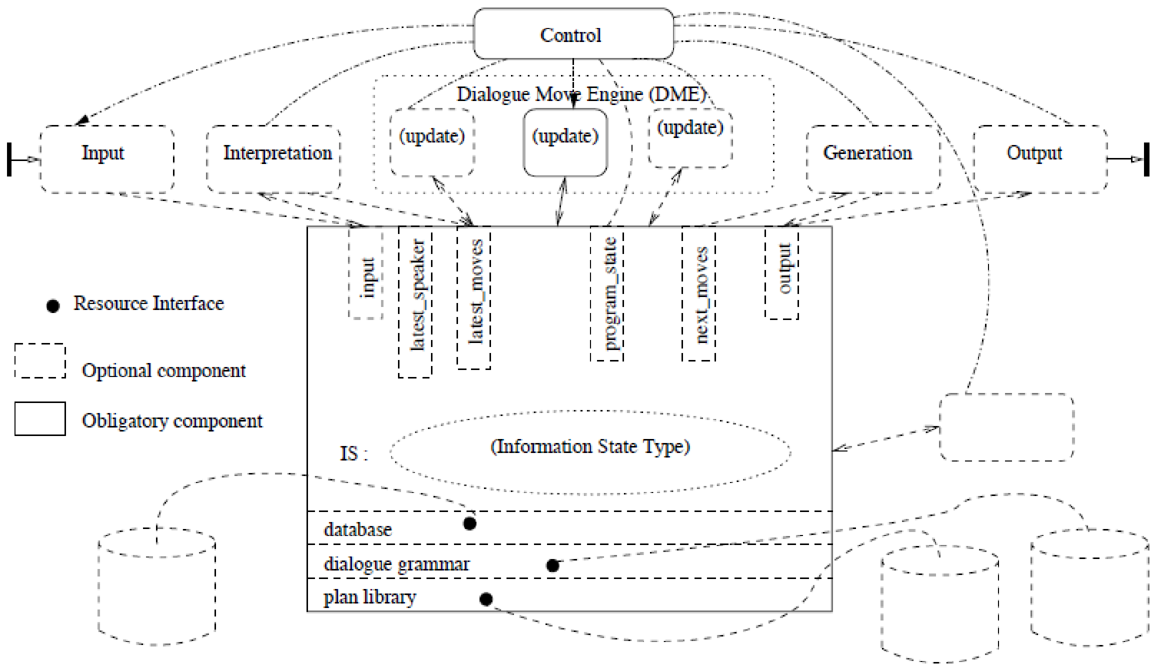


Figure 2.6: The TRINDI DME architecture, taken from (Larsson and Traum, 2000).

2.4.2 Rasa

Rasa (Bocklisch et al., 2017) was introduced in 2017 as an easy to use tools for building conversational systems and is being used by thousands of developers worldwide. Rasa is composed of a pair of tools: Rasa NLU (natural language understanding) and Rasa Core (dialogue management). Rasas' architecture was designed to be modular, i.e., Rasa NLU and Rasa Core are fully decoupled, allowing both Rasa NLU and Rasa Core to be used independently of one another. This also means that both can easily be integrated with other systems (e.g., Rasa Core can be used with NLU services other than Rasa NLU). The code is implemented in Python and both services can expose HTTP APIs to easily be integrated into projects that use different programming languages.

Architecture wise, the dialogue state is saved in a tracker object that is unique per conversation session. The tracker stores slots, as well as the log of all events that led to each state making each state able to be reconstructed by replaying all events.

Figure 2.7 shows how a user message is processed when received by Rasa.

1. The input message is received and passed to an *Interpreter* (Rasa NLU or another NLU tool) to extract the intent, entities and any other structured information;

2. The *Tracker* maintains the conversation state, and is notified when a new message is received;

3. The *Policy* receives the current state of the *Tracker*;

4. The *Policy* chooses which *Action* to take next according to the current state;

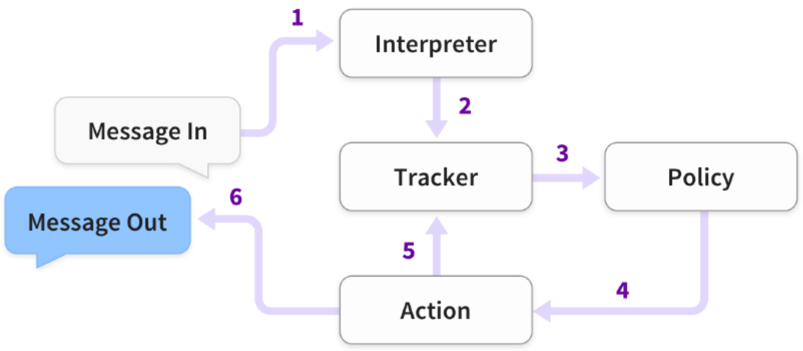


Figure 2.7: The general process flow of Rasa, taken from (Bocklisch et al., 2017).

5. The chosen *Action* is logged by the *Tracker*;

6. The *Action* is executed, which can result in sending a message to the user.

At each iteration, Rasa Core predicts which action to take: it can go from sending a simple utterance to the user, to execute a function. Whenever an action is executed, an instance of the tracker is passed as an input so it can make use of any relevant information collected over the history of the dialogue, i.e., slots, previous utterances, and the results of previous actions.

The policy is used to select the next action given the present tracker object. It is instantiated along with a *featurizer* that concatenates features of the current dialogue state:

- The last action;
- The extracted intent and entities from the most recent user input;
- Currently filled slots.

One recent system that uses Rasa is EvaTalk (Andrade et al., 2020), a chatbot system developed to attend users from *Escola Virtual de Governo* (EV.G), a Brazilian virtual school that hosts free and open courses. The development and implementation of this chatbot expected lowering the demand for the human support necessary to run the EV.G platform.

2.4.3 OpenDial

OpenDial (Lison and Kennington, 2016) was released as an open-source toolkit for building and evaluating spoken dialogue systems. OpenDial uses an information-state architecture (section 2.3.5) where the dialogue state is represented as a Bayesian network. Furthermore, the dialogue state is a shared resource for every module of the system.

With OpenDial, the domain models are specified via probabilistic rules encoded in a XML file.

2.4.3.1 Architecture

Figure 2.8 illustrates the general architecture of OpenDial.

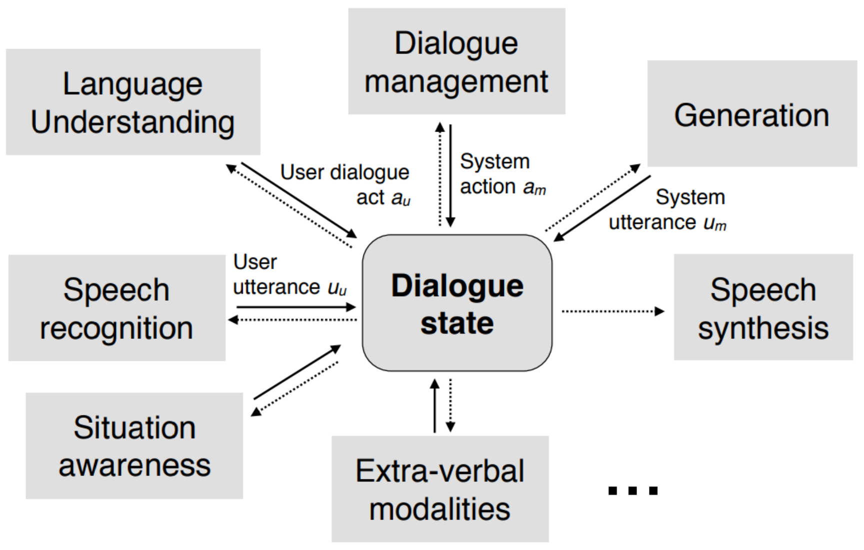


Figure 2.8: The OpenDial architecture with the dialogue state as a central shared memory, taken from (Lison and Kennington, 2016).

The dialogue state is divided into distinct variables, each of which representing an aspect of the human-computer interaction, such as:

- The user intention;
- The dialogue history;
- The external context.

This dialogue state is encoded as a Bayesian network which is a directed graphical model where the nodes represent the state variables and the edges represent conditional dependencies.

A typical information flow goes as follows:

1. The *Speech Recognition* module captures the user utterance u_u;
2. The *Language Understanding* module maps u_u into representations of dialogue acts, a_u;
3. For each a_u, the *Dialogue Manager* selects which action a_m the system must perform;
4. If a_m is a communicative act, *Language Generator* finds a linguistic realisation u_m;
5. The utterance u_m is sent to the *Speech Synthesiser*.

2.4.3.2 Domain

OpenDial is domain-independent. This means that any dialogue domain encoded in XML can be used with OpenDial. The XML domain file must contain information about:

- The initial dialogue state;

- A collection of domain modules composed of probabilistic rules;
- Prior distributions for unknown parameters in the probabilistic rules;
- Optional configurations settings.

The probabilistic rules in the domain models are expressed as `if...then...else` instructions that map logical conditions on some state variables to probabilistic effects on some other state variables. An example of a probabilistic rule within the XML file description is:

```
<if var="X" relation="in" value="[first,second,third]"/>
```

These rules are expressed as logical formulae, using logic operators (i.e., conjunctions, disjunctions, and negations) and binary relations (i.e., equality, inequalities, string matching, etc.). An example of a simple probabilistic rule is:

$$\forall x,\ \texttt{if}(a_u\ \texttt{= Request(x)} \wedge a_m\ \texttt{= Verify(x)) then}$$
$$\{\texttt{P(}a_u\texttt{' = Confirm(x)) = 0.9}$$

The example above expresses a prediction on the future user dialogue act a_u' based on the last user dialogue act a_u and system action a_m. The rule also specifies that the user is expected to confirm the request x with a probability of 0.9, when asked if the request was indeed x.

The `if...then...else` structure of the probabilistic rules divides the state space into groups of similar states (logical conditions). This enables developers to write rules with "backoff strategies", starting from the most specific condition and then gradually moving to more generic cases if the top conditions do not apply. This also comes important to ensure that the probabilistic rules are able to generalise to new and unseen situations.

2.4.4 PyDial

Released in 2017, PyDial (Ultes et al., 2017) is an open-source end-to-end spoken dialogue system toolkit that provides statistical approaches to all the components that make up a dialogue system, developed by the Cambridge University Engineering Department. PyDial supports multi-domain applications making it possible for a single conversation to have several different topics.

Figure 2.9 shows the general architecture of PyDial.

- The *Agent* is the main component and resides at the core of the system. It is responsible for the dialogue interaction since it encapsulates all the dialogue system modules to enable the interaction. The *Agent* pipeline contains:
    1. The *Semantic Decoder*, which transforms textual input into a semantic representation;

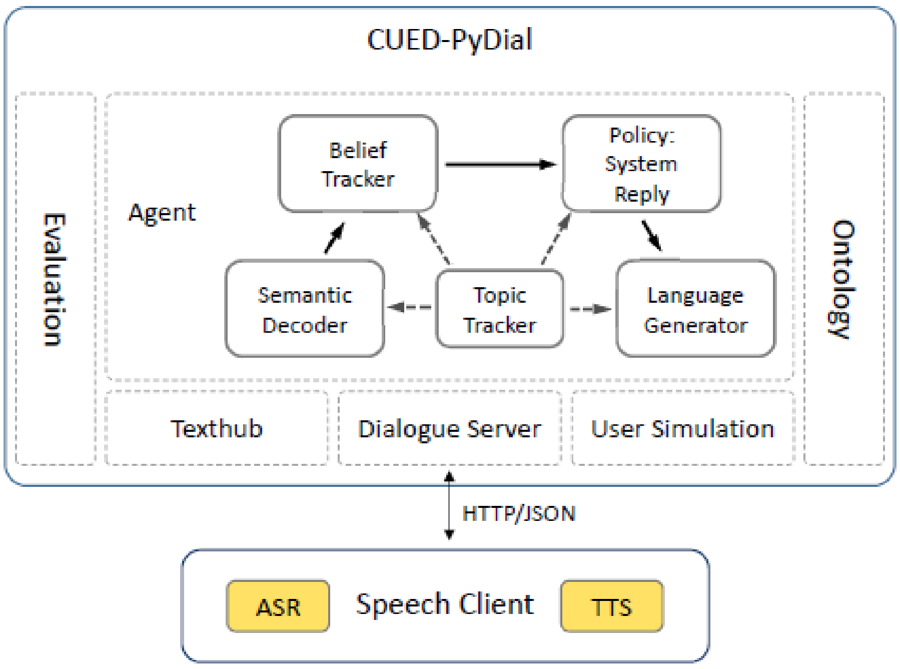


Figure 2.9: The general architecture of PyDial, taken from (Ultes et al., 2017).

2. The *Belief Tracker*, which is responsible for maintaining the internal dialogue state (the *belief state*);

3. The *Policy*, which maps the belief state to a suitable dialogue act;

4. The *Language Generator*, which transforms the system dialogue act to a textual representation;

5. The *Topic Tracker*, which is responsible for recognizing the context of the dialogue in multi-domain systems.

- The *User Simulation* provides the simulation of dialogues on the semantic level (see section 2.3.4), providing an approximation to the real user;

- The *Text Hub* connects the *Agent* to a terminal allowing it to communicate with text;

- The *Dialogue Server* enables speech-based dialogue with an external speech client.

- The *Ontology* specifies the dialogue domain and the access to the back-end database, for example, a set of restaurants and their properties;

- The *Evaluation* computes the evaluation measures for the dialogues, as explained in section 2.3.4.

2.4.5 ICECAPS

Microsoft ICECAPS (Shiv et al., 2019) (Intelligent Conversation Engine: Code and Pre-trained Systems), released in 2019, is an open-source natural language processing repository. It allows the usage of trained neural networks models for natural language processing, built on top of TensorFlow functionalities. Furthermore, this conversation modeling toolkit also emphasizes on multi-turn conversations while being able to maintaining the flow of the conversation and its context.

Regarding the architecture, ICECAPS was designed in a modular, easy to use and flexible way. Additionally, ICECAPS provides several built-in modules and configurations, such as:

1. *Personality grounding* using sequence-to-sequence models and transform decoders inspired by (Li et al., 2016);

2. A *Space Fusion* (Gao et al., 2019) preset that extends its multi-task capabilities. *Space Fusion* constructs a multi-task environment of two sequence-to-sequence models with a shared decoder (Luan et al., 2017) that regularizes several terms. These terms encourages responses for the same context to be placed nearby in latent space. This induces a structure in the latent space such that the distance and direction from a predicted response vector correspond to relevance and diversity. An example can be visualized in figure 2.10;

3. An extension of stochastic answer networks (Liu et al., 2018), a machine reading comprehension system, that acts as a full *knowledge-grounded* conversational model (Qin et al., 2019). This approach hybridizes machine reading comprehension with a response generation model.

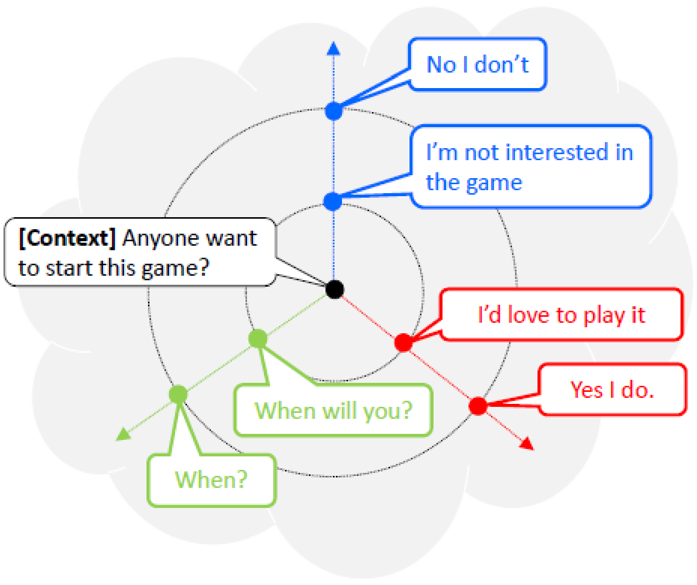

No I don't
I'm not interested in the game
[Context] Anyone want to start this game?
I'd love to play it
Yes I do.
When will you?
When?

Figure 2.10: Illustration of latent space under SpaceFusion, taken from (Gao et al., 2019)

2.4.6 Emora

The Emora STDM (State Transition Dialogue Manager) (Finch and Choi, 2020) is a dialogue management framework, released in 2019, offering the developer dialogue management techniques using state machines and information state. It also contains a natural language understanding module that works based on the analysis of inputs with regular expressions (NATEX). Information about both dialogue management and natural language understanding can be found on the following subsections.

2.4.6.1 NATEX

Emora presents the NATural language EXpression (NATEX) to address the understaning of natural language user inputs. It defines a comprehensive grammar to match patterns in user inputs by compiling them into regular expressions. Some of its most important features are:

- String Matching: The NATEX

  ```
  [I {watched, saw} $MOVIE = {Avengers, Star Wars}]
  ```

  matches user inputs such as `"I watched Avengers"` or `"I saw Star Wars"` and returns the variable `$MOVIE` with the values `"Avengers"` and `"Star Wars"` respectively;

- Function Calls: NATEX allows external function calls that are defined in Python. For example, the NATEX

  ```
  [I {watched, saw} $MOVIE = #MDB()]
  ```

  makes a call to the function `#MDB` that returns a set of movie titles. These functions can, for example, serve as database queries.

- Ontology: NATEX supports ontology editing and querying by calling the `#ONT` function. An ontology can be built either outside or inside the code of the system, but has to be loaded in JSON. `#ONT(movie)` searches for the node movie in the ontology and the NATEX

  ```
  [I {watched, saw} $MOVIE = #ONT(movie)]
  ```

  returns a set of movies from its subgraph;

- Response Generation: NATEX can generate system responses by randomly selecting one production of each disjunction (represented by {}). The NATEX

  ```
  I watched lots of $GENRE = {action, horror, drama} movies {recently,
  lately}
  ```

  can generate `’I watched lots of action movies lately’` or `’I watched lots of drama movies recently’` and assign the values of `’action’` and `’drama’` to the variable `$GENRE`.

- Error Checking: NATEX uses the Lark (Grosch, 2002) parser to automatically detect syntax errors. Furthermore, several types of error checking are performed before runtime, such as:
  - Calls to a non-existing function;
  - Exceptions raised by any function;
  - Functions returning mismatched type;
  - References to a non-existing variable.

2.4.6.2 Dialogue State Machine

The state machine component built within Emora STDM has states transitions that alternate between the user and the system to track turn taking and are defined with NATEX. A dialogue graph with a state machine approach is shown in figure 2.11.

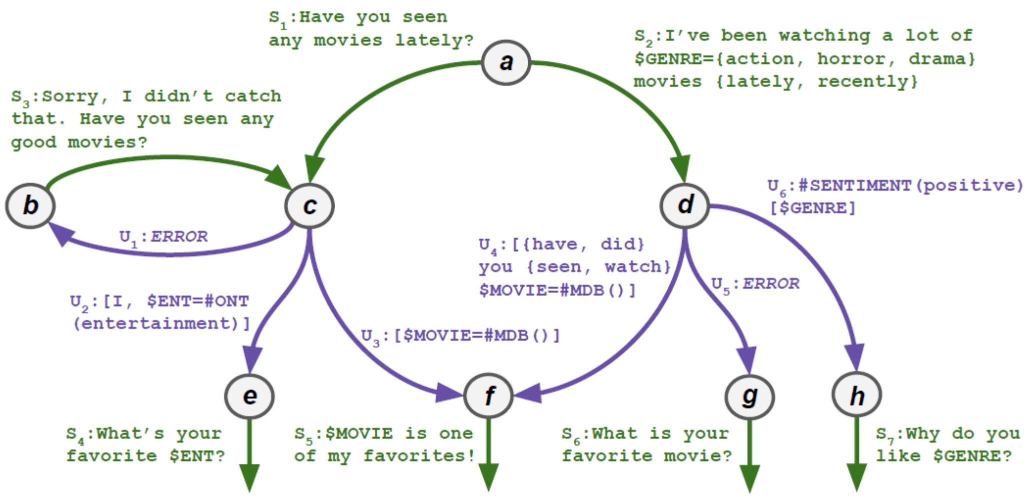

S1:Have you seen any movies lately?
S2:I've been watching a lot of $GENRE={action, horror, drama} movies {lately, recently}
S3:Sorry, I didn't catch that. Have you seen any good movies?
a
b
c
d
U1:ERROR
U6:#SENTIMENT(positive) [$GENRE]
U4:[{have, did} you {seen, watch} $MOVIE=#MDB()]
U5:ERROR
U2:[I, $ENT=#ONT (entertainment)]
U3:[$MOVIE=#MDB()]
e
f
g
h
S4:What's your favorite $ENT?
S5:$MOVIE is one of my favorites!
S6:What is your favorite movie?
S7:Why do you like $GENRE?

Figure 2.11: Dialogue graph using a state machine approach with NATEX to dialogue management, taken from (Finch and Choi, 2020)

User turns are modeled by transitions according to which NATEX matches the user input. In order to prevent multiple NATEX matches transitions can be defined with priority values.

System turns are modeled by randomly selecting an outgoing system transition for one specific user input. This promotes uniqueness in the dialogue making the conversation feel more natural. To avoid redundancy when returning to a specific state that has already been visited, Emora STDM prefers system transitions that have not been taken recently.

2.4.6.3 Information State Update Rules

Alongside state machine-based dialogue management, Emora STDM also allows developers to take advantage of information state-based dialogue management.

This approach contain two parts: a precondition and a postcondition. Each user turn before Emora STDM's state machine takes a transition, the set of update rules is evaluated with the user input until a candidate system response is generated or no rule's precondition is satisfied. For example in:

```
"[I have $USERPET = #PET()]":  "#ASSIGN($USERLIKE = $USERPET)"}
```

satisfying the precondition `"[I have $USERPET = #PET()]"` triggers the postcondition `"#ASSIGN($USERLIKE = $USERPET)"` to assign `$USERPET` to `$USERLIKE`, allowing another rule, for example `"#IF($USERLIKE != None)":  "I like $USERLIKE too!"`.

2.4.7 Plato

Plato (Papangelis et al., 2020) is a very recent tool (released on January 22th 2020) that can be used to create conversational agents, supporting interactions through speech, text or dialogue acts.

An application in Plato is composed by four major components:

- The *dialogue* which defines and implements dialogue acts and states;
- The *domain* which includes the ontology of the dialogue and the database that the dialogue system queries;
- The *controller* which orchestrates the conversations;
- The *agent* which implements different components of each conversational agent.

These components are depicted in figure 2.12, and figure 2.13 represents the flow of the interaction between user and system.

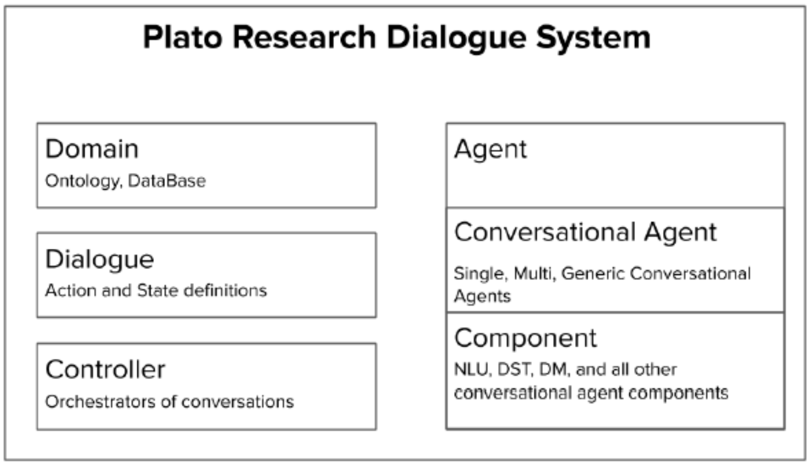

Plato Research Dialogue System
Domain
Ontology, DataBase
Dialogue
Action and State definitions
Controller
Orchestrators of conversations
Agent
Conversational Agent
Single, Multi, Generic Conversational Agents
Component
NLU, DST, DM, and all other conversational agent components

Figure 2.12: Major components of Plato, taken from (Papangelis et al., 2020).

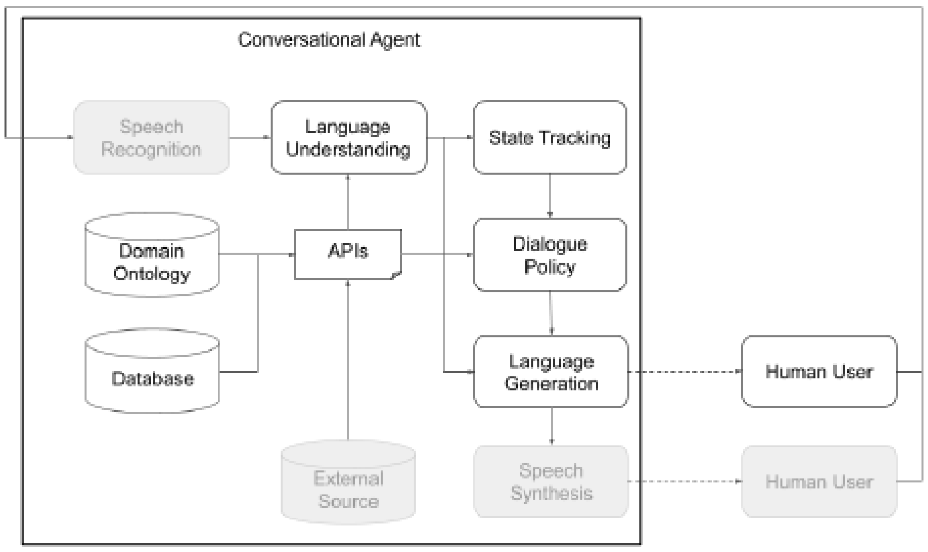


Figure 2.13: Flow of the human interaction with Plato, taken from (Papangelis et al., 2020).

2.4.7.1 Dialogue

Plato facilitates conversations between agents (i.e., user-system or simulated user-system) by making use of dialogue states and dialogue acts. These dialogue acts and states are formulated as soon as the module *Language Understanding* and are data objects that are passed throughout all the modules.

2.4.7.2 Domain

To implement a slot-filling task-oriented dialogue system in Plato, two elements that form the domain need to be specified: the *ontology* and the *database*. The *ontology* determines the informable slots, requestable slots, and system requestable slots for the conversation. The *database* contains the items that can fill the slots defined in the *ontology*.

The process of generating the *ontology* (a .json file) and the *database* (SQLite) is automated with Plato defined commands. The *database* is generated from a .csv file, with columns representing item attributes and rows representing items. An example of a database for a flower shop is presented in table 2.9:

Table 2.9: Example of a csv table that generates a database for a flower shop.

id	type	color	price	occasion
1	rose	red	cheap	any
2	rose	white	cheap	anniversary
3	rose	yellow	cheap	celebration
4	lilly	white	moderate	any
5	orchid	pink	expensive	any
6	dahlia	blue	expensive	any

2.4.7.3 Controller

The controllers are responsible for orchestrating the conversations between the agents. It instantiates them for each dialogue, keeping track of statistics and passing inputs and outputs properly.

The controller is defined in a configuration file, detailing the agents involved in the dialogue, as well as paths to the ontology and the database of the dialogue, along with other parameters and options (e.g., specifying who has the initiative, user or system).

2.4.7.4 Agent

Each application in Plato can have one or more agents. They are composed by a role (system or user) and a set of components such as natural language understanding, dialogue manager, dialogue state tracker, policy, natural language generator and user simulation. A conversational module containing these components can be divided into two different types: *rule-based* or *trained* modules.

A rule-based module works through slot filling (slot filling NLU, DST, policy, NLG), following rules and/or patterns defined on the ontology and database.

Plato allows the training of components either online (during the interaction) or offline (through logs).

2.5 Critical Analysis and Preliminary Evaluation of Tools

The final section of this chapter presents the analysis and comparison of the tools presented throughout the previous section. Based on this comparison, the selection of tools used in this dissertation is justified.

When exploring tools to use on the systems presented on this document, three main points were kept in mind:

- Portuguese support;

- Ease of implementation;
- Development with few data;
- Modularity and allow extensions with other tools;
- The cost associated to each of them.

Table 2.10 presents a comparison between the studied frameworks, presenting what each of them was developed aimed at.

Table 2.10: Comparison of the features available in the presented frameworks. SM: state machine, IS: information state, ON: ontology, ML: machine learning support.

Framework	License	Language	SM	IS	ON	ML
Rasa	Commercial	Python	✓			
OpenDial	MIT	Java		✓		
PyDial	Apache 2.0	Python		✓		
ICECAPS	MIT	Python				✓
Emora STDM	Apache 2.0	Python	✓	✓	✓	
Plato	Apache 2.0	Python			✓	✓

Emora (Finch and Choi, 2020) was chosen as the framework to handle the dialogue management, as it allows the development of both Information State and State Machine dialogue managers, while providing developers an initial easy to learn approach to the framework. Moreover, Emora does not give restrictions regarding the language that the systems are aimed towards. The main restriction however, is the lack of modularity presented by the framework, which motivates the improvements presented in chapter 3.

To handle the task of understanding inputs and generating outputs, Plato (Papangelis et al., 2020) was chosen due to the presented capacities to develop slot filling modules as well as trained modules using machine learning techniques. This means that NLU and NLG modules can be developed with few data (slot filling), with the possibility of extending the module features with lots of data (machine learning).

CHAPTER 3

Emora+ Framework

Even though Emora (Finch and Choi, 2020) system has the tools to develop dialogue managers as is, some additional features were considered necessary in order to develop more structured and versatile systems. This chapter presents information regarding the changes and improvements made to the core Emora framework to make it more suitable to the objectives detailed in the following chapters. Furthermore, it also explains the reasoning behind the integration of Plato (Papangelis et al., 2020) into Emora: it gives the capacity of developing more capable and sophisticated NLU modules.

First and foremost, section 3.1 presents the improvements and changes made to Emora as well as developments targeting making the process of creating and developing new Emora based dialogue systems easier. After, section 3.2 introduces how Plato's features can be utilized alongside dialogue managers developed with Emora. Finally, the overall system architecture reflecting the integration of all the improvements previously mentioned is presented followed by an overview of the main steps required to use Emora+ to create a dialogue system.

3.1 Improvements to Emora

While maintaining the integrity of the source code, the developed features allowed an improvement to how the interaction between user and system works, making it possible for the user to use the system beyond than just a terminal (command line). Furthermore, the new functions aim at providing developers with additional support when developing new systems. These functions are the following:

1. Decouple the state-machine from the intrisic need of a command-line, enabling, for example, remote browser based interactions and making possible the connection of the Emora based dialogue manager with other frameworks' NLU and NLG modules;

2. Preserve information regarding interactions with the system;

3. Provide the developer improved information for debugging stages;

4. Support of on-boarding by making possible to know the current state and the number of previous state machine visits to that state;

5. Simplify the creation of the initial Emora skeleton for new domains.

These features will be explained in detail on the following sections.

3.1.1 State Machine Decoupling

The main limitation that was identified, when experimenting with Emora, was that the only way a user can interact with a system is through the command line, which is not ideal in a real case scenario. Furthermore, the only way that the state machine could transition between states was through user inputs on the same command line. This meant that the system state would be lost anytime the command line closed, and restarted from scratch anytime the program was executed.

Since the systems developed on the scope of this project had in mind an intuitive and user-friendly human-machine interaction, the previously mentioned issues had to be overcome.

The main addition to Emora was a new method that allows the state machine to run in the background and have the transitions between states be triggered by a function call, in an API based approach. This is also the change that makes Emora a modular framework, allowing the integration of more advanced NLU and NLG modules with Emora based dialogue managers. In this regard, the importance and impact of this novel approach, is explained better in sections 3.2 and 3.2.1.

The presented solution is to change the way an Emora based system runs when executed. In the current approach, when a system is executed, after defining the states and transitions, the system enters in a *run* method that is infinitely awaiting a user input in the command line.

When an input is inserted, the state machine receives the input and executes a *user_turn* method. This method processes the user input, triggering the correct state transition. If the new state corresponds to a system state, it sets the *SPEAKER* global variable to the system. Since the new speaker is the system, an immediate *system_turn* is executed that runs similarly to *user_turn*.

The proposed approach, in order to enable state machine decoupling, consists in the creation of a new method named *service_run* that replaces the entire original *run* Emora method, meaning that a system no longer gets locked in an infinite *run* call.

This allows the state machine definition code to be imported by a front-end interface, and be initialized by the call of a starting method that defines:

- The ontology file - the file that defines the concepts that the system can process;

- The user and system states - corresponding to each state that the system can be at a given instant;

- The macros - can be seen as functions that can be called by the system;

- The regular expressions - which define what triggers the transitions between states;
- The transitions between states - the definition of the transitions associates two states and what triggers the transition between one and another (regular expression).

Whenever a new user input needs to be processed, the interface sends it to the state machine definition code, where the new *service_run* method is called, returning to the interface the system response. The *service_run* method executes a *user_turn* with the received input and returns the correspondent *system_turn*. With *service_run*, the system runs in the background, and the user never interacts with it directly, but with an interface that calls methods to the dialogue manager definition.

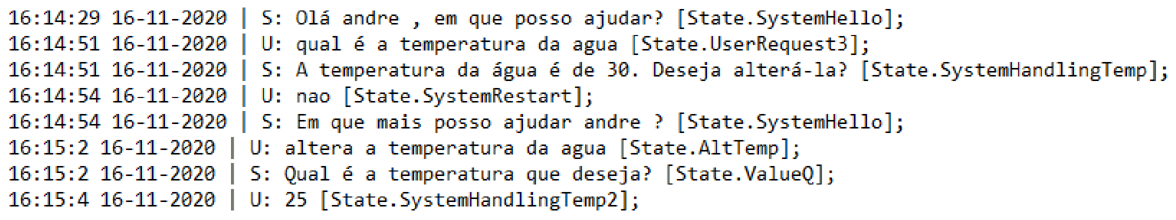
3.1.2 Logging

Logging is a useful feature used in many systems that helps developers find problems during and after the deployment of their applications, easing the process of localizing problems.

With a new logging method, the developer can now save the logs of the interaction between user and system. The new *set_log_file* method requires a name as input, sets a global variable, *logging*, to true and names the log file after the input it receives. It automatically generates a file on a specific folder for logs only, where the name of each file is defined by the developer.

To keep this process simple and well structured for the developer, it is recommended that the name of each log file has a direct relation to the system user, to ensure that the developer can, later, study what each specific user intended to use the system for as well as what they were expecting the system to be able to do.

An example of a log file is presented in figure 3.1. Note that the log is in Portuguese, as the specific system was developed to recognize and generate Portuguese inputs and outputs.



```
16:14:29 16-11-2020 | S: Olá andre , em que posso ajudar? [State.SystemHello];
16:14:51 16-11-2020 | U: qual é a temperatura da agua [State.UserRequest3];
16:14:51 16-11-2020 | S: A temperatura da água é de 30. Deseja alterá-la? [State.SystemHandlingTemp];
16:14:54 16-11-2020 | U: nao [State.SystemRestart];
16:14:54 16-11-2020 | S: Em que mais posso ajudar andre ? [State.SystemHello];
16:15:2 16-11-2020 | U: altera a temperatura da agua [State.AltTemp];
16:15:2 16-11-2020 | S: Qual é a temperatura que deseja? [State.ValueQ];
16:15:4 16-11-2020 | U: 25 [State.SystemHandlingTemp2];
```

Figure 3.1: Example of a piece of a log file of a Portuguese interaction.

As seen in figure 3.1, the file follows a defined and specific structure, saving information such as:

- The time when each interaction took place;
- An indication saying if it is a user or system turn;
- The text of each interaction;
- The name of the system state that the system was currently in.

Note that each information is intentionally separated with specific symbols. The reason behind this strategy is to make it possible to use information retrieval procedures to extract specific information of the log file. One important example is extracting user's turns to further train a NLU module developed with Plato, improving the process of understanding future user inputs.

3.1.3 Additional Debugging Information

When developing and testing a system of this nature, every possible input needs to be tested and the system adjusted accordingly. To help the developer on this task, a new method was proposed to provide the developer with the information needed when debugging a new system. This information is provided to the developer when interacting with it via command line in the form of a dictionary, and it ranges from:

- The last transition performed by the system, i.e., the last and current state;
- What the system understood from the input and caused the transition;
- A list of possible transitions from the current state;
- The name of the system state that the system is currently in.
- The number of times the system has visited the current state.

This additional debugging information is an add-on to the debugging provided by the logs, serving as a different level of debugging, since, for example, it can be accessed in runtime.

3.1.4 On-boarding Support

One cannot guarantee that when a user uses a system, it will be intuitive enough for the user to not require help.

With the new information that can be gathered thanks to the debugging features described in section 3.1.3, developers can apply on-boarding techniques in dialogue managers developed with Emora. These on-boarding techniques allow the developer to provide users with inputs aiming to help them use the system.

If a user cannot perform an action, after several unsuccessfully recognized inputs, the developer can program the dialogue manager to provide extra information to the user that can help him perform said action. Another example, is the possible necessity for the system to be more helpful to the user when reaching a specific state for the first time. Both of these examples make use of the track kept for the number of times a specific state has been visited.

Figure 3.2 represents an approach on how on-boarding can be handled. When the system is on the `ASK_REQUEST` state, i.e., asking the user the desired request, it can transition into two different states: `ERROR` or `PROCESS_REQUEST`. If the system is not capable of processing the request, it transitions into the `ERROR` state where the *err* variable is incremented. When the *err* variable is not equal to 3, the system transitions back to the `ASK_REQUEST` state, with the same flow. When the system transitions into the `ERROR` state for the third time in a row, it resets the *err* variable and transitions into the `INITIAL` state, providing the user, for example, a different output as to which types of request it can process.

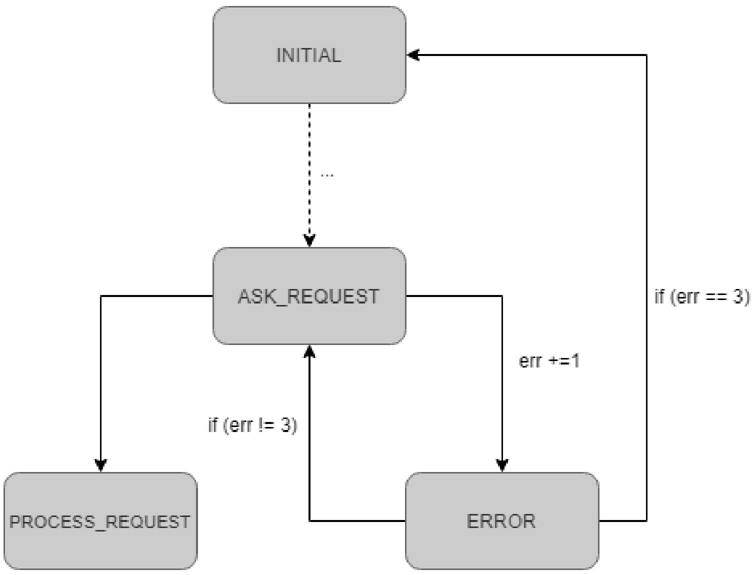


Figure 3.2: Illustration of an example of an on-boarding approach: when the system cannot process the user request three times in a row, it provides the user a different output and transitions back to the initial state.

3.1.5 High Level Tools to Support Developers

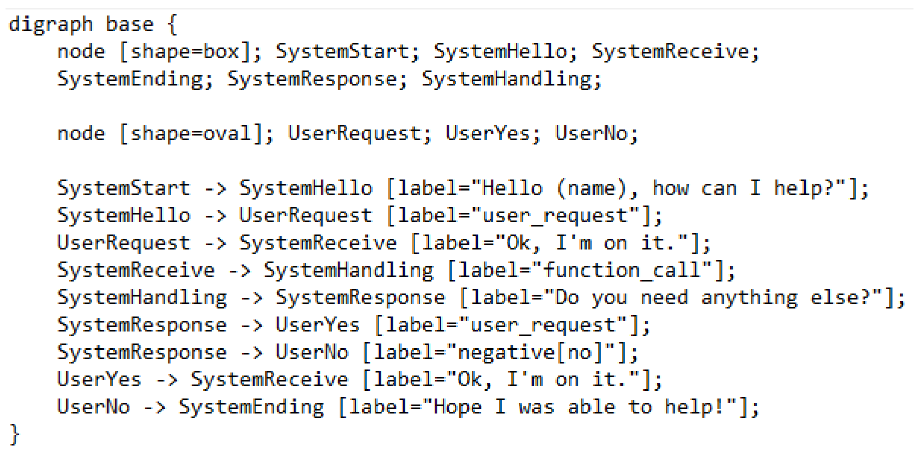
To ease the process of creating a new Emora based system from scratch and since they are based in state machines, a script that converts a visual representation of the state machine into the programming skeleton of the Emora system was developed.

For the visual representation, Graphviz (GraphViz, 1991) was selected, an open source graph visualization software capable of representing structural information such as diagrams of abstract graphs and networks. It transforms a programmed representation of a graph into a visual representation of it, following certain rules.

A state machine can be defined, in Graphviz, adopting the following rules:

- System states are defined as "boxes";
- User states are "oval";
- Transitions between states are defined with a "->";
- The label defined in each transition defines the trigger of that specific transition;
- Ontology elements can be marked in the label by enclosing them in square brackets.

Figure 3.3 shows a simple example of the code required to specify a particular graph and figure 3.4 shows the correspondent graph.



```
digraph base {
    node [shape=box]; SystemStart; SystemHello; SystemReceive;
    SystemEnding; SystemResponse; SystemHandling;

    node [shape=oval]; UserRequest; UserYes; UserNo;

    SystemStart -> SystemHello [label="Hello (name), how can I help?"];
    SystemHello -> UserRequest [label="user_request"];
    UserRequest -> SystemReceive [label="Ok, I'm on it."];
    SystemReceive -> SystemHandling [label="function_call"];
    SystemHandling -> SystemResponse [label="Do you need anything else?"];
    SystemResponse -> UserYes [label="user_request"];
    SystemResponse -> UserNo [label="negative[no]"];
    UserYes -> SystemReceive [label="Ok, I'm on it."];
    UserNo -> SystemEnding [label="Hope I was able to help!"];
}
```

Figure 3.3: Simple example of a definition of a graph using Graphviz.

All five of the mentioned rules can be identified in both figure 3.3 and 3.4: the system states are represented with rectangles (boxes) and the user states are represented with an oval; the transition between two states is represented with a "->" and have a label that defines what triggers that transition; the transition between states *SystemResponse* and *UserNo* also define an ontology element, defining that "no" is one way of referring to the camp "negative".

The developed script processes the textual definition of the graph (figure 3.3) and generates the skeleton of the state machine definition file. If the script recognizes any ontology element, it also automatically generates the ontology JSON file. Then, it does the following:

1. Includes the ontology file path;
2. Lists both user and system states;
3. Initializes specific system requirements;
4. Defines the state machine transitions, taking into account if the starting state is a user or system state, with the specified label as the trigger.

Figure 5.4 presents the resulting state machine definition file and figure 3.6 the ontology file, using the developed script and taking the textual definition of the graph as an input. The script is used with the following command:

```
python3 script.py [input_file].gv [output_file].py [ontology_file].json
```

where `input_file.gv` is the file that contains the textual definition of the graph, `output_file.py` contains the state machine definition, and `ontology_file.json` contains the definition of the system ontology.

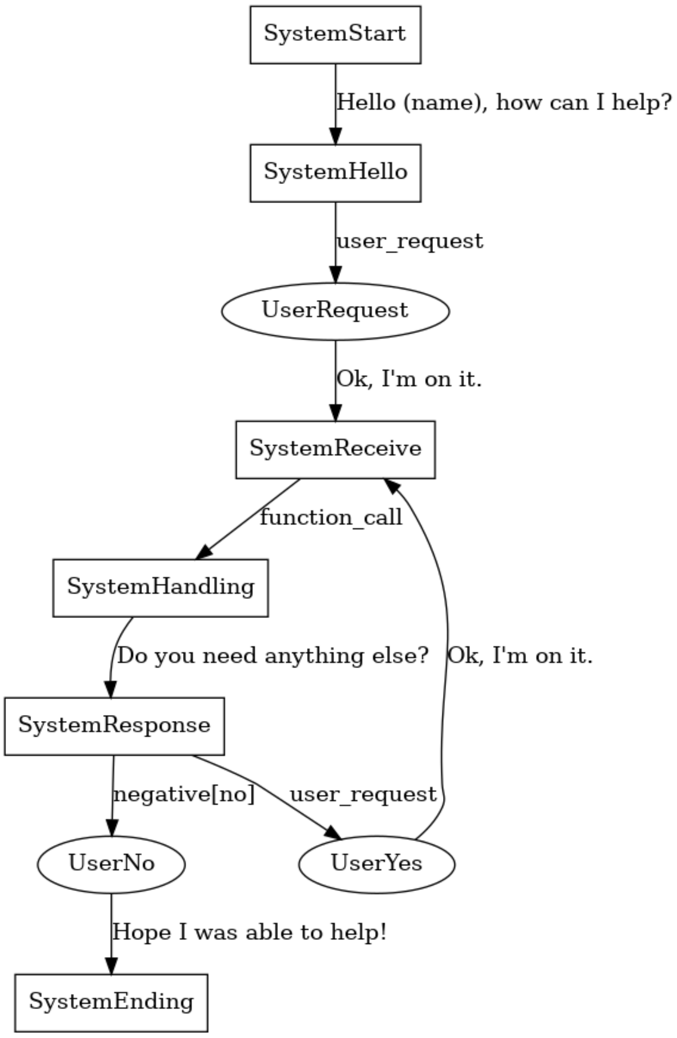


Figure 3.4: Simple example of a graph generated by Graphviz.

```
# AUTO GENERATED FILE
from emora_stdm import DialogueFlow, Macro, KnowledgeBase, NatexNLG
from enum import Enum, auto
import json

# WARNING: complete ontology file found at example.json
ontology = {}

with open('example.json') as json_file:
     ontology = json.load(json_file)

class State(Enum):
     # system states
     SystemStart = auto()
     SystemHello = auto()
     SystemReceive = auto()
     # user states
     UserRequest = auto()
     UserYes = auto()
     UserNo = auto()

knowledge = KnowledgeBase()
knowledge.load_json(ontology)
df = DialogueFlow(State.START, initial_speaker =
DialogueFlow.Speaker.SYSTEM, kb = knowledge)

df.add_system_transition(State.SystemStart, State.SystemHello,
'"Hello (name), how can I help?"')
df.add_system_transition(State.UserRequest, State.SystemReceive,
'"Ok, I'm on it."')
df.add_system_transition(State.UserYes, State.SystemReceive,
'"Ok, I'm on it."')
df.add_system_transition(State.SystemHello, State.UserRequest,
'"user_request"')
df.add_system_transition(State.SystemResponse, State.UserYes,
'"user_request"')
df.add_system_transition(State.SystemResponse, State.UserNo,
'"negative[no]"')
```

Figure 3.5: Generated state machine definition file of the presented example, using the developed script.

```
{
    "ontology": {
        "negative": [
            "no"
        ]
    }
}
```

Figure 3.6: Generated ontology definition file of the presented example, using the developed script.

3.2 Integration of Plato functionalities

As explained in section 2.5, Plato was chosen alongside Emora to develop a new dialogue system framework, and with the improvements explained throughout section 3.1 an improved framework arised, Emora+.

The integration of Plato into Emora is justified with one main aspect: the capability that Plato has to develop more capable and sophisticated NLU modules, with either intent recognition or even with trained modules using neural networks.

The issue that could arise is the lack of training data to develop such neural network strategies, but the development of the logging feature in section 3.1.2 ensures that the interactions between users and the system can be extracted from the logs and be used as training data for the trained NLU module.

Furthermore, Plato has improved characteristics to give the user more suitable answers to each specific input, and as so Plato's NLG module is also integrated into Emora.

With the addition made to Emora, regarding debugging features, as explained in section 3.1.3, a Plato agent can serve as the interface and initialize an Emora based dialogue manager. When Plato receives a user input, the NLU module extracts the important information from the input and the agent calls the relevant Emora method to process the extracted information. The dialogue manager transitions the state, giving the Plato agent the system answer allowing the NLG module to form a suitable answer to the user.

The following subsection presents the final architecture for Emora+, explaining this process in more detail.

3.2.1 General Architecture

Figure 3.7 shows the resulting architecture of the improvements made to Emora as well as the integration with Plato.

In the figure, modules in white correspond to Plato while the dark grey one corresponds to Emora. Furthermore, both ASR and TTS are surrounded by a dashed rectangle as they are optional: these two modules are only used if the interaction between user and system is through voice and not by text.

The user utterance U_t is received by Plato's NLU that extracts from U_t the intents and dialogue acts U_d that are provided to Emora's dialogue manager.

The dialogue manager takes actions according to U_d and sends the systems response S_d to Plato's NLG. The latter is responsible of adapting S_d into a user-friendly answer S_t.

If the interaction is through voice, Plato's ASR module is responsible of transforming the user input audio U_s into a text representation U_t that is sent to the NLU. Similarly, Plato's TTS module transforms the system response in text form S_t into a sound representation S_s.

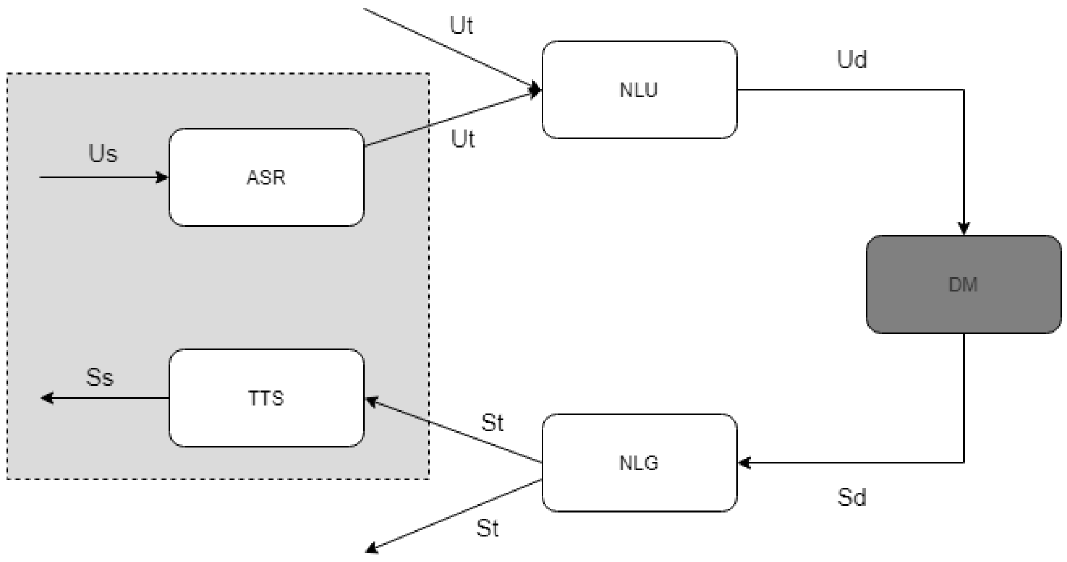


Figure 3.7: The general architecture of the Emora+ framework. Modules in white correspond to Plato and dark grey corresponds to Emora. Modules surrounded by a dashed rectangle are optional. ASR: Automatic Speech Recognition, NLU: Natural Language Understanding, DM: Dialogue Manager, NLG: Natural Language Generator, TTS: Text-to-Speech.

3.3 Procedures To Create a Dialogue System

The recommended brief procedure to create a dialogue system with Emora+ is the following:

1. Sketch and develop the initial graph in Graphviz, in iteractions until it feels like a good system base;
2. Use the script to automatically generate the ontology file as well as the system description file;
3. Complete the ontology file with desired concepts;
4. Iteratively test and complete the dialogue manager with more state possibilities;
5. Define the Plato NLU module to use:
    - Slot-filling is ready to go;
    - For trained NLU modules, use the logs of the interactions with the system used for the testing of the dialogue manager as training data.
6. Improve and test the NLG module until it gives desirable answers to the user.
7. Develop a user-friendly frontend for the user to interact with the developed system.

3.4 Overall Conclusions

Additional features were considered necessary in order to develop more structured and versatile systems with the frameworks selected (see section 2.5). While maintaining the integrity of the frameworks' source code, the developed features allowed an improvement to how the interaction between user and system works aim at providing developers with additional support when developing new systems. Furthermore, the addition of Plato (Papangelis et al., 2020) to Emora (Finch and Choi, 2020) gives capabilities to develop more capable and sophisticated NLU modules, with either intent recognition or even with trained modules using neural networks.

The resulting improved framework allows the development of two systems presented in the following chapters.

CHAPTER 4

Dialogue System for a Smart Home

This chapter presents a proof-of-concept developed with the knowledge presented in chapter 2 and the framework developed in 3. This proof-of-concept was developed in the scope of the Smart Homes topic, and it was developed taking in mind the previous work developed in (Ketsmur et al., 2019).

First are presented the context and the environment onto which the system was idealized and designed to, followed by the scenario containing examples of dialogues. The requirements presents the devices that compose the scenario and what they must be able to do. Following the requirements, the development section presents and details the steps that were taken when developing this system. Finally, the last section of this chapter presents the results of the system, enforced with evaluations from real life users.

4.1 Context

When developing any kind of system, the developers must always take into account its purposes and the environment that it will be inserted in. The proof-of-concept system presented in this chapter was designed and developed in alignment with the Smart Green Homes project, a joint project between University of Aveiro and Bosch, specifically aimed at a pseudo T0 implemented on the UX (User Experience) space at Bosch Thermotechnology in Aveiro.

Since the system development had a T0 apartment in mind, the divisions of the house are limited to a kitchen, a living room, and a technical room. Each of these rooms is equipped with a few devices that can provide information about their status (e.g. water heater) and/or can be controlled by the user (e.g., Lights).

Table 4.1 associates house divisions to every device that it contain:

As explained further ahead, the system must also provide the user with information about consumptions regarding the household. This means that three different resources can be used by the equipments presented in the table: electricity, water and gas.

Table 4.1: Overall context considered for the Smart Homes proof-of-concept, aimed at a three rooms apartment with several devices.

Division	Devices
Kitchen	GEOS (water heater) EWI/Smart plug Dishwasher Lights
Living Room	Air distribution (air purifier) Sensor box (indoor air quality) Lights Television (presents information to the user) Tablet (multimodal interaction with user)
Techinal Room	Air purifier HP (heat pump) CDI Softner (conductivity sensor) Home server

4.2 Scenario

The scenario corresponds to a demonstration of the smart home and represents the home return of a married couple after a day at work.

The owners of the smart home just got home, and the husband enters the living room, where he:

Scene 1 - Light intensity adjusting

Turns on the illumination of the space, adjusting its intensity, by interacting with the system through voice.

Scene 2 - Consulting and adjusting of the ambient temperature

Interaction with the smart home system using a tablet to consult and change the ambient temperature.

Scene 3 - Consulting the temperature and quality of the water and adjusting its temperature

The wife wants to take a shower, and to not waste time, she uses the system with her voice to interact with the system, and through the dialogue with the assistant she:

- Consults the current temperature of the water;
- Adjusts the temperature of the bath water;

- Consults information regarding the quality of the water.

Scene 4 - Air quality alert

Through a notification on the tablet, the assistant transmits an alert regarding the air quality. Upon reading this notification on the tablet, the husband uses the assistant to activate the Air Purifier.

Scene 5 - Turning on dishwasher and water quality alert

The husband goes to the kitchen, and through a voice interaction with the assistant, the dishwasher is turned on. After turning on the dishwasher, the assistant immediately sends a notification to the husband warning that the water quality is bad, asking him if he wants to turn on the CDI softner. Using a simple on/off button, the husband turns on the CDI softer in order to avoid the use of additives.

4.3 Requirements

Different devices in different divisions compose the household. Naturally, each device has its own functionalities that must be described.

Table 4.2 presents information regarding each of the devices mentioned on the previous section. This table contains details about the type of queries allowed for each device, as well as what type of actions they allow and what alerts they must provide to the user. The presented devices can be turned on/off, except the *Sensor Box* that alerts the user regarding the quality of the air. Beyond just turning on/off devices, some allow other queries such as consulting the specific device consumption history.

Table 4.2: List of devices and functionalities in the Smart Home.

	Query	Control	Alert
GEOS	Consumption history State (on/off) Errors + Error solutions	Turn on/off Set water temperature Increase/descrease the water temperature	
EWI/Smart Plug	State (on/off)	Turn on/off	
Dishwasher	Consumption history State (on/off) Errors + Error solutions	Turn on/off	
Washing Machine		Turn on/off	
Air Purifier	Consumption history State (on/off) Errors + Error solutions	Turn on/off	
Sensor Box	Current and history of air quality		Air quality
Lights	Consumption history State (on/off)	Turn on/off Set and increase/decrease intensity	
Television	Consumption history State (on/off)	Turn on/off	
Heat Pump (HP)	Consumption history State (on/off) Errors + Error solutions Current and past temperatures	Turn on/off Set and increase/decrease temperature	
CDI Softner		Turn on/off	

4.4 Development

The development phase of the dialogue system for a smart home presented in this chapter can be divided into different steps:

1. Initial sketch of the dialogue system containing some dialogue possibilities;

2. Populating ontology based on the requirements and scenario;

3. Improvements to the dialogue manager by iteratively testing;

4. Integration with a back-end system;

5. Development of a user-friendly web-based front-end.

The work carried out to accomplish each of these steps is explained and summarized in the following sections.

4.4.1 Initial Dialogue Fluxes

An initial sketch of the dialogue manager was created using a simple drawing tool. This first sketch intended to represent some of the initially conceptualized features of the system and the correspondent dialogue possibilities serving as grounds for discussion and improvement. The result is depicted in figure 4.1.

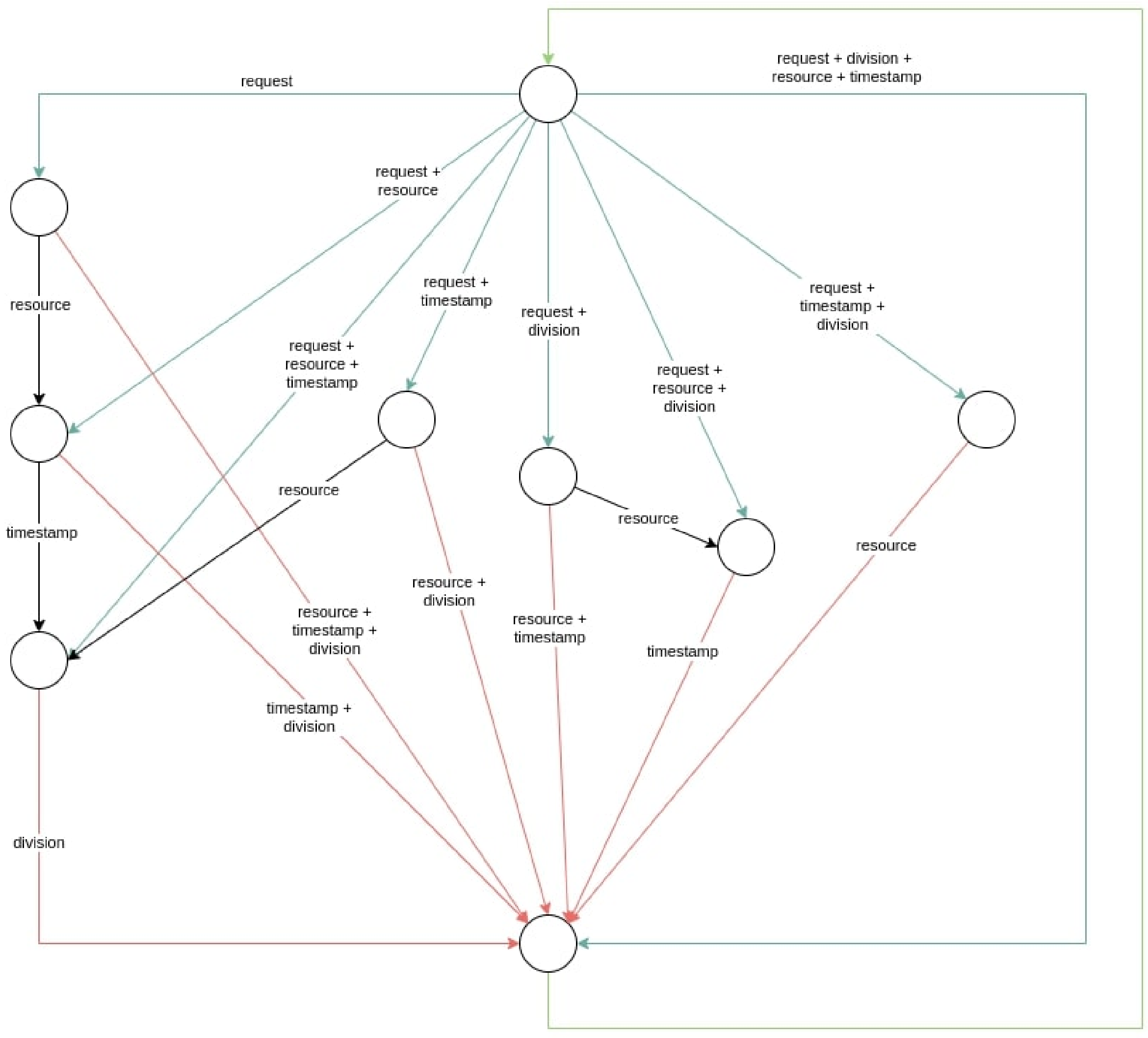


Figure 4.1: The initial smart home dialogue system diagram, developed using draw.io. States are represented as circles and the arrows between them are the transition triggers.

The circles on the figure represent different states that compose the system, while the arrows represent the transitions between states. The label of the arrows correspond to inputs from the user which trigger transitions. The uppermost central state is the initial state of the system, whereas the bottom-most is the state correspondent to the execution of the user inputs.

This figure is a very early concept of a system that could provide the user information regarding resource consumption on the house. Taking the leftmost path as an example, if the system recognized a *request* from the first user input, it then would require the *resource*, a *timestamp* and the *division* from the user to complete the final request. The system also allows the user to provide several slots of information in just one input, even accepting a user input containing all the information it needs to execute the requested command (e.g., the rightmost path).

This sketch was then extended and migrated to Graphviz, to enable a more programmable definition of the diagram and a later processing of it, as described in section 3.1.5, generating the skeleton of the dialogue system definition file. The result is the diagram in figure 4.2.

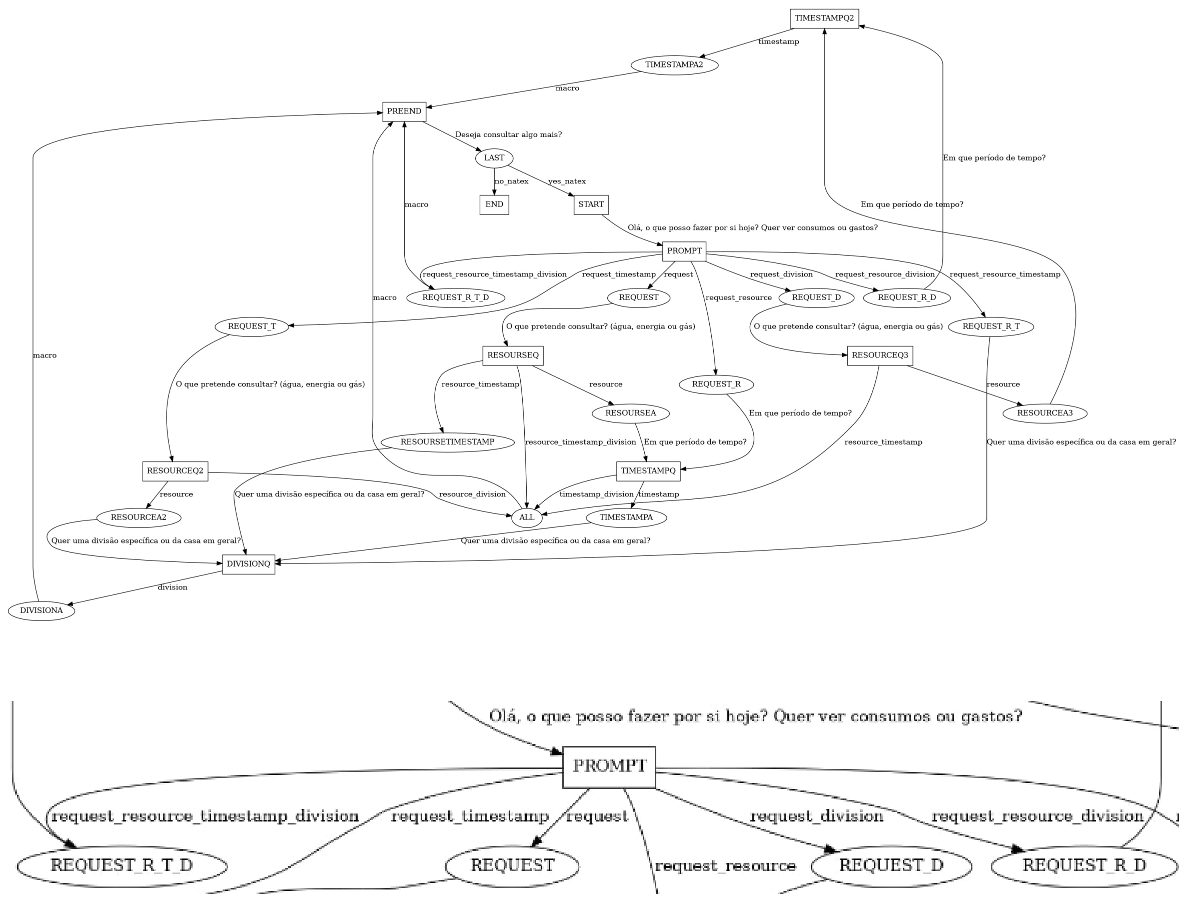


Figure 4.2: On top, a smart home dialogue system diagram, developed using Graphviz. System states are represented as boxes (rectangles) and the user states are represented as ovals. The arrows between them are the transition triggers and can contain ontology elements. On the bottom, a detailed view of part of the diagram to better depict it.

The resemblance between figure 4.2 and figure 4.1 can be seen in the way that the transitions are defined: a user input makes the system transition between the initial state `PROMPT` to the state that corresponds to such input. Furthermore, this diagram is a closer approach, compared to the latter, to how dialogue managers are defined with the user framework, as system states and user states are specified differently. As explained in 3.1.5, in this diagram system states are represented as boxes (rectangles) while user states are represented as ovals.

Figure 4.3 is a graphical representation of the scenes presented in 4.2, with a closer look in scene 3.

Defining possible dialogue fluxes for the considered scenes helped defining states that were needed, as well as elements of the ontology.

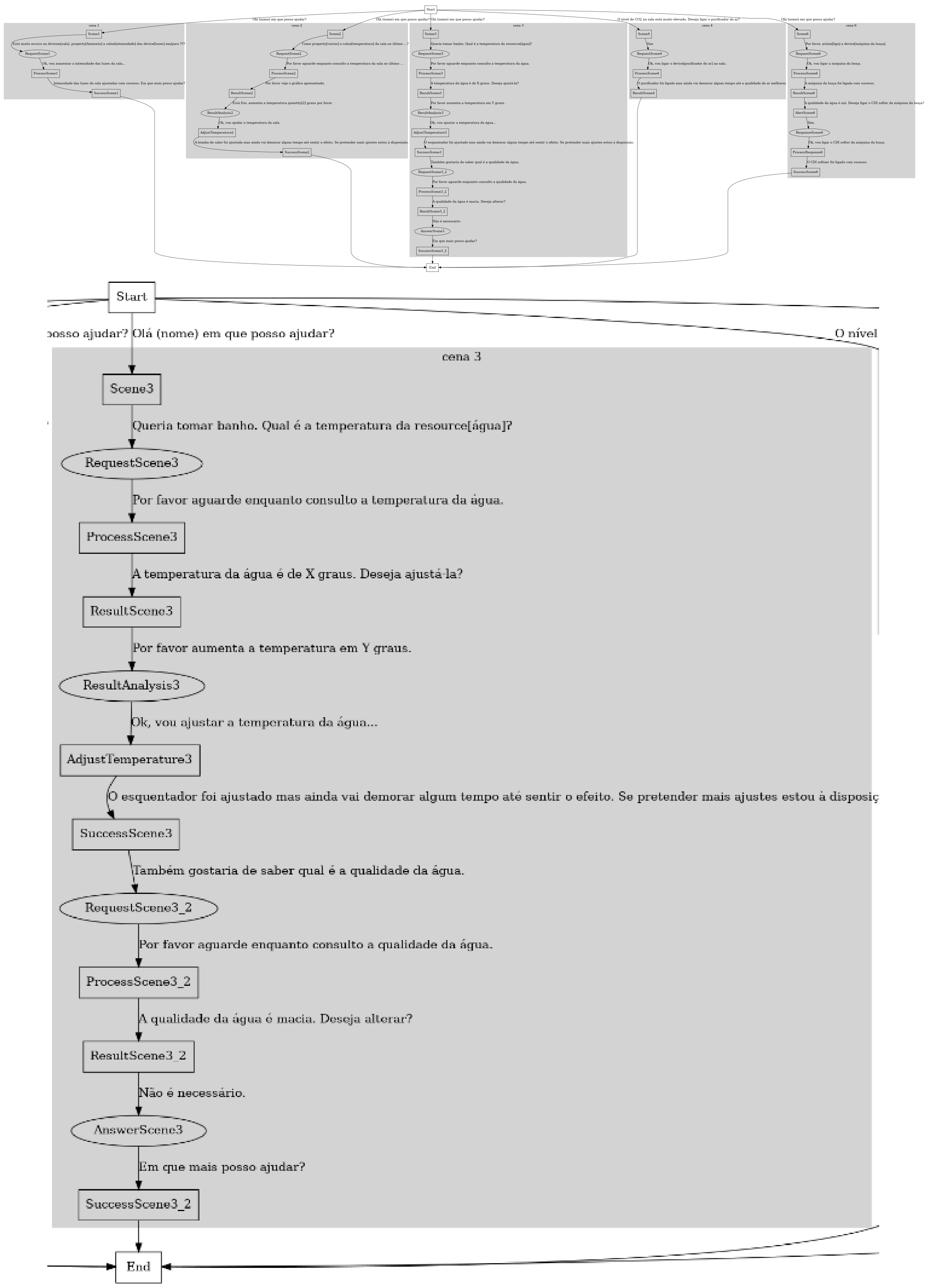

Start
Olá (nome) em que posso ajudar?
O nível
cena 3
Scene3
Queria tomar banho. Qual é a temperatura da resource[água]?
RequestScene3
Por favor aguarde enquanto consulto a temperatura da água.
ProcessScene3
A temperatura da água é de X graus. Deseja ajustá-la?
ResultScene3
Por favor aumenta a temperatura em Y graus.
ResultAnalysis3
Ok, vou ajustar a temperatura da água...
AdjustTemperature3
O esquentador foi ajustado mas ainda vai demorar algum tempo até sentir o efeito. Se pretender mais ajustes estou à disposiç
SuccessScene3
Também gostaria de saber qual é a qualidade da água.
RequestScene3_2
Por favor aguarde enquanto consulto a qualidade da água.
ProcessScene3_2
A qualidade da água é macia. Deseja alterar?
ResultScene3_2
Não é necessário.
AnswerScene3
Em que mais posso ajudar?
SuccessScene3_2
End

Figure 4.3: Graphical representation of Scenario. On top are presented the five different scenes. On the bottom, a detailed view of part of the diagram, corresponding to scene 3.

4.4.2 Completing the Ontology

The ontology represents the knowledge base of the system, i.e., the definition of what the dialogue system will be able to understand.

Using the state diagrams depicted in figures 4.2 and 4.3, presented on the previous section, and the script to convert the former into an Emora system skeleton, (see section 3.1.5), the first dialogue system description file was the result, as well as the correspondent ontology file.

Since this specific dialogue system was developed in the scope of smart homes, the system ontology must, naturally, contain information regarding them. An initial theoretical ontology helped understanding entities that must be integrated in the system ontology. These include:

- House divisions;
- Devices that can be controlled in the house;
- Resources used in the house;
- Different types of requests that the user can make.

Using the definition of the requirements and scenarios, as well as an idealized theoretical ontology, the system ontology could be completed. Table 5.4 is used to explain more in detail some specific entities that are part of the ontology. Classes like `Division` and `Device` refer to house elements that the assistant has to be capable of processing. `Request`, `Timestamp` and `Quality` are elements that, together with other classes like the aforementioned `Device`, allow the system to process full user requests: for example, the element *temperature* of the class `Quality` is a property of the element *water* from class `Resource`, making the assistant capable of processing a user input like `"please change the temperature of the water"`.

Table 4.3: Explanation of some examples of elements from the system ontology.

Class	Description	Entity example
Division	Part of the house	Cozinha (kitchen)
Device	House device	Televisão (television)
Request	Request regarding a device and/or division	Liga (turn on)
Resource	Resource that the house uses	Água (water)
Timestamp	More complete requests	Último mês (last month)
Quality	Resource characteristics	Temperatura (temperature)
Affirmative	Affirming	Sim (Yes)

Figure 4.4 shows the definition of how the system ontology is specified with an example of a defined class (*device*) in Portuguese.

Although the development of an ontology wasn't the main focus of this work, it was a resource that had to be tackled to enable building the actual system. To this end, the proposed

```
{
    "ontology": {
        ...
         "device": [
             "GEOS",
             "aquecedor de agua",
             "aquecedor de água",
             "luzes",
             "roupa",
             "louça",
             "louca",
             "purificador",
             "televisão",
             "televisao",
             "tv",
             "tablet"
         ],
         ...
    }
}
```

Figure 4.4: Example taken from the JSON system ontology. The figure presents the class `"device"` which contains the devices considered in the Smart Home.

ontology was designed to contain everything deemed necessary for the proof-of-concept without it being too extensive.

4.4.3 Iterative Improvement

Having the more complete ontology and the initial skeleton, the next step was to iteratively test, debug and improve the dialogue system by:

- Adding more states and state transitions, using for example the graphical representation of the scenes (figure 4.3);
- Giving the dialogue system the functionalities that refer to the defined and considered devices;
- Allowing distinct ways to complete a request, i.e., considering several ways to say one thing: for example, "change" and "alter" are considered the same request;
- Integrating the backend developed in (Ketsmur et al., 2019) which makes the assistant capable of providing the user with resource consumption.

4.4.4 Integration with Back-end

As previously mentioned, one of the main goals of this proof of concept was to take advantage of a previously built system that allows the retrieval of information regarding house consumption, and can be considered as the back-end and database of the full system.

The database contains simulated consumption for several devices and divisions, and the data is aggregated into different timestamps, allowing for consumption requests for, for example, one specific month or for the last week.

The queries to the database are in SPARQL, and different steps are taken to translate the user input into the database SPARQL query:

1. The dialogue manager receives user intents, in this case, regarding house consumption;
2. When it has all the information needed, via one or more turns, it formulates the final query;
3. The dialogue system queries the SPARQL database and receives the correspondent information;
4. The dialogue system formulates the answer to the user with the consumption information.

4.4.5 Front-end

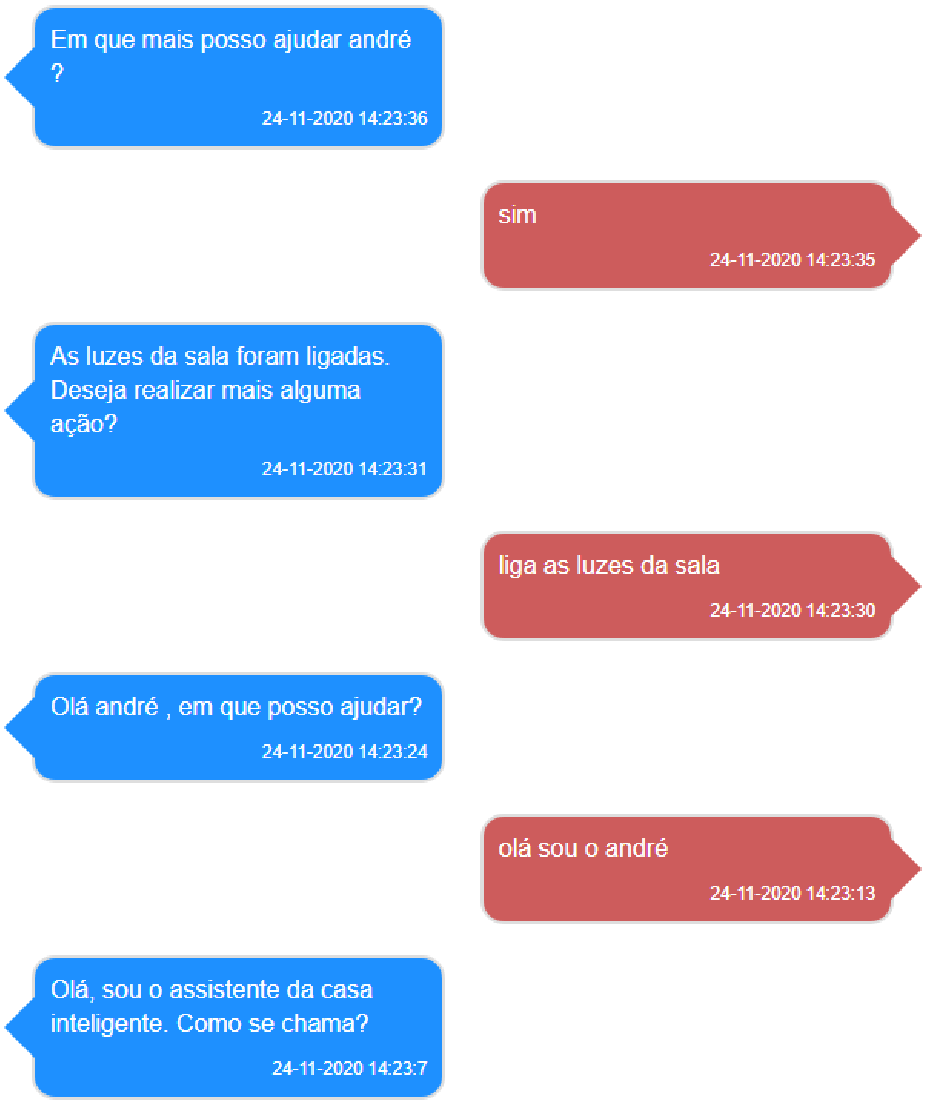
The last step was to develop a user interface than enabled user to experiment with the system, as debugging and testing was made through the command-line. The system itself is stored on a remote computer, and it can be accessed remotely [1]. To make the interaction between user and system more fluid and friendly, a front-end based on Flask was developed.

[1] `http://aal.ieeta.pt/assistant_static/index.html`

Flask is a framework for web development applications using POST and GET methods, which fits perfectly into the needs of the system.

The front-end itself is an approach to a very common concept of chats, where the user inserts the command and to which the system responds. When the front-end receives an input, it makes calls like an API to the dialogue manager (see section 3.1.1)

Figure 4.5 shows an example of usage of the system.



Em que mais posso ajudar andré ?
24-11-2020 14:23:36

sim
24-11-2020 14:23:35

As luzes da sala foram ligadas. Deseja realizar mais alguma ação?
24-11-2020 14:23:31

liga as luzes da sala
24-11-2020 14:23:30

Olá andré , em que posso ajudar?
24-11-2020 14:23:24

olá sou o andré
24-11-2020 14:23:13

Olá, sou o assistente da casa inteligente. Como se chama?
24-11-2020 14:23:7

Figure 4.5: An example of the usage of the system with the developed front-end. Boxes in red correspond to the user input, while boxes in blue correspond to the system output.

4.5 Results

The assistant presented in this chapter was subjected to evaluations by real users.

This section summarizes the essentials of the results. First are presented representative examples of interactions withdrawn directly from the addition of the logging feature to Emora (section 3.1.2). Following these examples, are analysed the reactions and functioning of the assistant.

4.5.1 Examples

Example 1 - Consulting and changing the temperature of the water: The user wanted to know the water temperature and then change it.

```
14:42:46 2-12-2020 | S: Olá Diogo , em que posso ajudar? [State.SystemHello];
14:42:58 2-12-2020 | U: qual a temperatura da agua [State.UserRequest3];
14:42:58 2-12-2020 | S: A temperatura da água é de 20. Deseja alterá-la?
[State.SystemHandlingTemp];
14:43:3 2-12-2020 | U: sim [State.Value];
14:43:3 2-12-2020 | S: Qual é a temperatura que deseja? [State.ValueQ];
14:43:6 2-12-2020 | U: 26 [State.SystemHandlingTemp2];
14:43:6 2-12-2020 | S: A temperatura da água foi alterada para 26. Deseja
realizar mais alguma ação? [State.SystemHandlingTemp3];
```

After presenting to the user the temperature of the water, the assistant considered the possibility that the user would also want the change it.

Example 2 - Turning on the living room lights and the television: In a second example, the user was interested in turning on the living room lights and the television.

```
14:52:37 2-12-2020 | S: Olá Diogo , em que posso ajudar? [State.SystemHello];
14:52:48 2-12-2020 | U: podes ligar as luzes da sala [State.LightsSpace];
14:52:48 2-12-2020 | S: As luzes da sala já estão ligadas. Deseja realizar
mais alguma ação? [State.DeviceManagement];
14:53:1 2-12-2020 | U: liga a tv [State.ManageDevice];
14:53:1 2-12-2020 | S: A tv já está ligada. Deseja realizar mais alguma ação?
[State.DeviceManagement];
14:53:12 2-12-2020 | U: não [State.SystemEnding];
14:53:12 2-12-2020 | S: Espero ter sido útil! [State.SystemEnding];
```

The assistant correctly processed the user request, processing the device and the house division that the user was referring to.

Example 3 - Consulting the kitchen water consumption in the last month: In the final example, the user wanted to get information regarding the water consumption in the kitchen in the last month.

```
14:53:41 2-12-2020 | S: Olá Diogo , em que posso ajudar? [State.SystemHello];
14:54:11 2-12-2020 | U: qual o consumo de agua no último mês da cozinha
[State.RequestConsumo1];
```

```
14:54:11 2-12-2020 | S: Quer ver consumos de água, gás ou energia?
[State.ConsumoWhich];
14:54:16 2-12-2020 | U: agua [State.ConsumoThis];
14:54:16 2-12-2020 | S: Quer ver consumos da sala ou da cozinha?
[State.ConsumoWhere];
14:54:25 2-12-2020 | U: cozinha [State.ConsumoHere];
14:54:25 2-12-2020 | S: Em que período de tempo? [State.ConsumoWhen];
14:54:31 2-12-2020 | U: no último mês [State.ConsumoThen];
14:54:38 2-12-2020 | S: O consumo de agua na cozinha último mês foi de
1616.3 m3. Deseja realizar mais alguma ação? [State.CheckConsumo];
14:54:45 2-12-2020 | U: não [State.Ending];
```

Although the user entered a full request, the assistant was not capable of processing it entirely, only being able to process the part of the input regarding the request. The assistant then asks the user the missing information, before finally presenting the user the consumption value he was initially seeking.

4.5.2 Users' Evaluations

A group of people was asked interact with the assistant, after reading some information regarding it. It was asked the same users to evaluate the assistant on a score of 1 to 5, pointing out strengths and weaknesses.

Users were asked to explore the assistant's functionalities to the fullest while interacting with it in the most natural way possible. Users were not given a list of tasks to do, in order to not condition the users.

The test was carried out by a group of seven people, aged between 13 and 40 years old with degrees between the third basic cycle and doctorate. Furthermore, the users had different levels of knowledge regarding the assistant and the scenario.

4.5.2.1 Users' Evaluation Results

As previously mentioned, users were asked to, after using the assistant, evaluate it on a score of 1 to 5. Table 4.4 shows results regarding the evaluations provided by the users.

Table 4.4: Results of the users' evaluations.

Mean	STD	Median	Mode	Min	Max
3.5	0,76	4	4	2	4

In addition to evaluating the system with a score, each user also pointed out aspects that they considered positive and negative. Tables 4.5 and 4.6 presents information regarding the most referred positive and negative aspects. Table 4.6 also presents notes on how negative aspects could be corrected.

Table 4.5: Positive assistant aspects that users pointed out after using the assistant.

The assistant's ability to detect missing information and provide clues on how to complete the request.
It informs that a device is already turned on/off when the user asks to turn on/off a device that is already on/off.
After handling a request and providing the user the answer, the assistant always asks a follow up question so the conversation keeps flowing naturally.
Intuitive, adaptable and responsive suitable for a chat dynamic.
Reacts appropriately, when asking whether to take another action, to an answer "yes, [followed by a request]".

Using the logging method (see section 3.1.2), the interactions between users and the assistant can be processed. This makes it possible to understand the reasoning behind certain issues and fix them. Figures 4.6, 4.7 and 4.8 represent results regarding the processing of the extracted information from the generated logs of three different users.

The first analyzed log (refer to figure 4.6) showed three different situations: the assistant was capable of processing the user request (`OK`), the assistant was not capable of processing the request (`NOT OK`), or the assistant processed wrong the request (`WRONG`). The assistant processed the user input correctly 67% of the times, miss processed the request once and was not capable of processing 31% of the requests. Both when the assistant processed wrong the request or failed to process it, the problem was the NLU module. However, the assistant was capable of processing the input:

```
como foi o gasto de agua na última semana na cozinha
```

and responded

```
O consumo de agua na cozinha última semana foi de 0.0 m3.  Deseja realizar
                          mais alguma ação?
```

In this situation, while the NLU module was capable of processing the full request, the back-end failed to provide a valid value.

Beyond the three different situations described in the first log, the second analyzed log (figure 4.7) shows an extra one: the user entered a non valid input (for example, a blank input or an input that contained a character with decoding issues) (`BAD INPUT`). In this log, the assistant was only able to correctly process 32% of the requests, miss processing 3 times (7%) and not being capable of processing 57% of the requests. The same NLU problems were notorious when analyzing this log. However, to the input:

```
liga o tablet
```

the assistant responded with an empty answer. This means that the NLU was capable of processing the input, but the dialogue manager did not know what to do with such input.

Table 4.6: Negative assistant aspects that users pointed out after using the assistant. On the second column, observations are presented as to how these issues can be fixed.

Negative Aspects	Observations
Chat interaction should be done in reverse, not bottom-up but top-down.	This would be fixed with an alteration to the front-end and is not an issue associated with the assistant itself.
Empty outputs.	This is an issue associated with the Dialogue Manager: there might not have been a state associated to a certain input nor definition of what the assistant should do to said input.
Confirmation in information that is provided very clearly. For example, to the request "Turn on the kitchen lights" the assistant answered “The living room or kitchen lights?".	This is associated to the NLU: like in example 3 of section 4.5.1, the assistant was not capable of processing the entire request, only processing "turn on" and "lights".
Fails to process if information is provided in certain orders.	This is also an NLU issue that would be solved with data and trained NLU module.
Turning on/off the lights resulted in turning on/off the television.	This happened because someone else was using the assistant at the same time. The Dialogue Manager is initialized once a user connects to the assistant. However, if another user connects to the assistant and it is already running, the assistant loses track as to which information it has to provide to which user.

The final log (figure 4.8) presents the results extracted from the interaction of a third user. In this log, the assistant either processed the input correctly (`OK`) or failed to do so (`NOT OK`). This log has examples of most of the negative aspects presented in table 4.6. To the user request:

```
qual foi o consumo de agua no último ano
```

the NLU only processed the part of the input regarding consumption, asking the user which resource he wanted to check as well as in which division and time, when the resource and time was already provided in the first input. Furthermore, when asked the desired water temperature, the user answered:

```
cinquenta
```

and the NLU failed to process it as the ontology contained information regarding temperatures in numeral form. Finally, to the request:

```
sim liga as luzes da sala
```

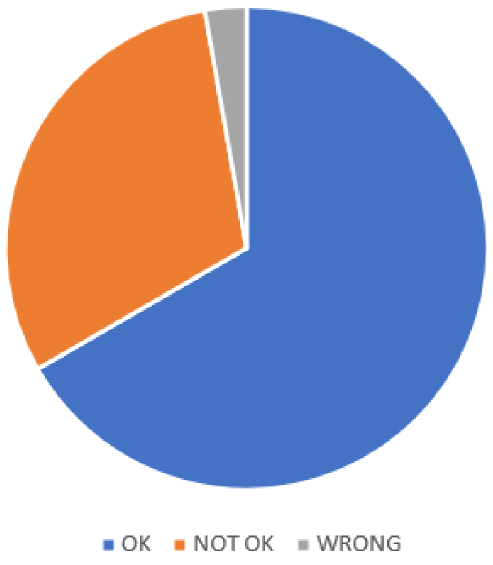


Figure 4.6: Graphical representation of the first user log. `OK` means the assistant was capable of processing the request, while `NOT OK` means that the assistant could not perform that task, and `WRONG` means it processed the request wrong.

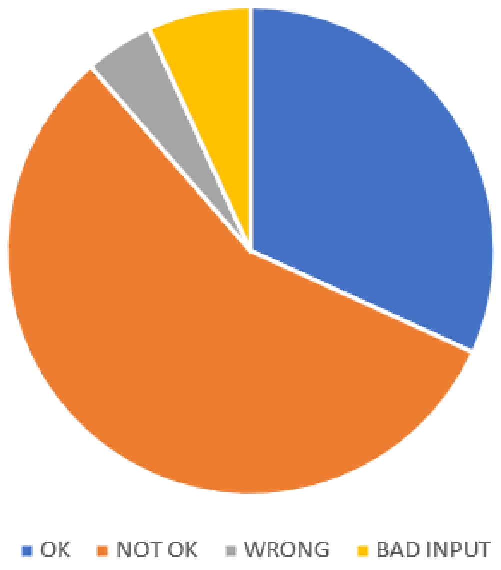


Figure 4.7: Graphical representation of the first user log. `OK` means the assistant was capable of processing the request, while `NOT OK` means that the assistant could not perform that task, `WRONG` means it processed the request wrong, and `BAD INPUT` means that the user entered a non valid input.

the assistant answered with a blank output. Although the author cannot say for sure, he believes that another user was using the system at the same time, making the assistant confused as to which information it had to provide to which user.

In this log, the assistant correctly processed the user input 63%, failing to do so in 37% of the user inputs.

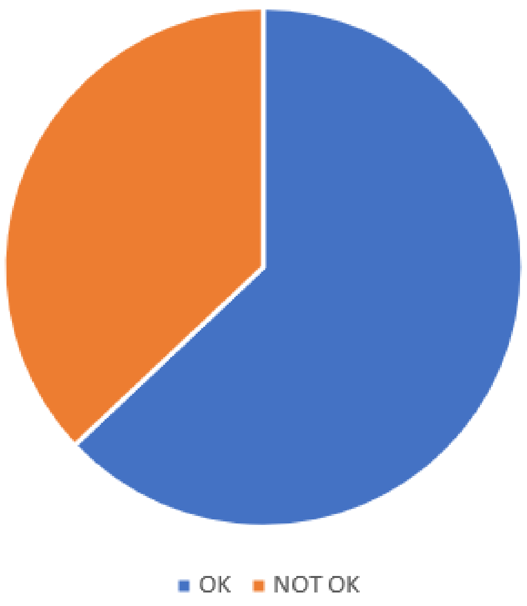


Figure 4.8: `OK` means the assistant was capable of processing the request, and `NOT OK` means that the assistant could not perform that task.

CHAPTER 5

Proof of Concept for Accessible Tourism

This chapter presents a second proof-of-concept developed with the knowledge presented in chapter 2 and the framework developed in 3. This proof-of-concept was developed in the scope of the accessible tourism topic.

Following this introduction, a scenario of a user interacting with the assistant is presented. Then, theoretical dialogue examples between a tourist and the assistant are also presented. Finally, the development section details the steps that were taken when developing this system.

5.1 Scenario

This specific system was developed having in mind a specific group of users: tourists with any kind of special need or disability. The presented scenario envisions a person with mobility limitations that wants to travel through Portugal.

Pedro is a young man who likes to travel, especially trips that allow him to get to know Portugal as much as possible. He has good computer skills and is used to looking for information online for his work and leisure.

He decides that he wants to visit Lisbon and accesses the website `www.turismoacessivel.pt` where he interacts with the assistant to obtain adequate information regarding his condition (he is paraplegic) for a trip to the Lisbon Oceanarium.

At another time, and due to a business meeting, Pedro has to visit Aveiro and uses the assistant again to obtain information regarding a cheap hotel that has wheelchair accesses.

The following section presents representative examples of the interaction between Pedro and the tourism assistant.

5.1.1 Dialogue Examples

Tables 5.1, 5.2 and 5.3 present 3 interaction examples (in Portuguese and in English). Words highlighted are terms that allow the understanding of the ontology necessary for a system of this nature.

The first example depicts a possible dialogue between Pedro and the assistant. Pedro intends to visit Lisbon and go to the Oceanarium, but needs to know before-hand if it has easy accesses because of his condition.

Table 5.1: First example of dialogue. The tourist intends to visit the oceanarium but needs to know if it has good accessibility preemptively, due to his condition.

TOURIST:	Estou a pensar ir a **Lisboa**.	I'm planning to visit Lisbon.
ASSISTANT:	Eu conheço algumas coisas sobre Turismo em **Lisboa**. O que pretende?	I know some tourism information in Lisbon, what do you intend to do?
TOURIST:	Gostava de ir ao **Oceanário**, mas tenho receio que com a minha **cadeira de rodas** seja complicado.	I would like to go to the oceanarium, but I'm afraid that it might be complicated due to my wheelchair.
ASSISTANT:	Backend - SPARQL query to Knowledge base...	
	Pode visitar o **Oceanário** sem problemas, é um **local** com **acessibilidade** muito **boa** para a sua **condição**.	You can visit the oceanarium without any issue, it is a point of interest with very good accessability for your condition.
TOURIST:	Isso é ótimo, vou então comprar bilhetes.	That is great, I will then buy some tickets.

The second example represents a possible dialogue resulting of the second part of the scenario. Pedro needs to visit Aveiro for a business meeting, and needs somewhere cheap with access for wheelchairs to spend the night.

The final example represents a dialogue between a tourist and the assistant. The tourist wants to travel with his family to Lisbon, and asks the assistant for suggestions regarding points of interests and hotels that are suitable for his blind son.

Table 5.2: Second example of dialogue. The tourist has a business meeting in Aveiro and needs to a room to spend the night.

TOURIST:	Tenho uma reunião de negócios em **`Aveiro amanhã`** e preciso de um sítio onde passar a **`noite`**.	I've a business meeting in Aveiro tomorrow and need a place to spend the night.
ASSISTANT:	Pretende passar a **`noite`** num **`hotel`** ou **`turismo rural`**?	Do you want to spend the night at an hotel or rural tourism?
TOURIST:	Estou à procura de um **`hotel barato`** e que permita **`acessos`** com **`cadeira de rodas`**.	I'm looking for a cheap hotel that allows wheelchair access.
ASSISTANT:	Backend - SPARQL query to Knowledge base...	
	O hotel X tem tudo o que precisa pelo preço Y.	Hotel X has everything you need for Y.
TOURIST:	Ok obrigado.	Ok thank you.

Table 5.3: Third example of dialogue. The tourist is spending a weekend with his family out and wants suggestions of what to visit.

TOURIST:	Quero passar um **`fim de semana`** com a minha família em **`Lisboa`**. Que tipo de **`pontos de interesse`** recomendas?	I want to spend the weekend with my family in Lisbon. Which points of interest would you recommend?
ASSISTANT:	Backend - SPARQL query to Knowledge base...	
	Em **`Lisboa`** existem vários **`pontos de interesse`** que lhe podem interessar como o **`oceanário`**, **`museus`**, **`parques`** e **`praias`**.	In Lisbon there are several points of interest that might interest you like the oceanary, museums, parks and beaches.
TOURIST:	O meu filho é **`invisual`** por isso esses pontos também têm que dar algo interessante ao meu filho.	My son is blind so those points of interest must give something to him as well.
ASSISTANT:	Backend - SPARQL query to Knowledge base...	
	O seu filho pode aproveitar a natureza tanto na **`praia`** como em **`parques`**. Para além disso, podem visitar o **`museu`** A.	Your son can enjoy the nature in the beaches and parks. Furthermore you can visit the museum A.
TOURIST:	Ok obrigado. Também preciso de **`dois quartos`** de **`sábado para domingo`**.	Ok thank you. I also need two bedrooms for saturday night.
ASSISTANT:	Backend - SPARQL query to Knowledge base...	
	O hotel X tem **`dois quartos`** disponíveis e tem bastante **`acessibilidade`** para **`invisuais`**.	Hotel X has two bedrooms available and has plenty of accessibility for blind people.
TOURIST:	Ok obrigado.	Ok thank you.

5.2 Development

Adopting the process refered in chapter 3, the development phase for the tourism assistant can be divided into smaller steps:

1. Definition of the scope of the system with Departamento de Turismo;
2. Definition of the domain ontology;
3. Populating a database with information;
4. Development of an initial base dialog system.

5.2.1 Domain Ontology for Emora+

The ontology of this system represents the knowledge base and what it will be able to process from user inputs. This specific dialogue system is aimed at accessible tourism, which means it must contain information about concrete entities that can be sub-categorized and classified. The considered entities are, amongst others, the following (Table 5.4):

Table 5.4: Explanation of some examples of elements from the system ontology.

Class	Description	Entity example
Location	Location containing information	Aveiro
Accommodation	Accommodations in a location	Hótel (Hotel)
Point of interest	Specific points of interest in a location	Museus (Museums)
Affirmative	Affirming	Sim (Yes)

Some of the ontology elements have properties that must also be defined: for example, a point of interest can have a timetable and a price. These properties are defined on a database. As an example, Table 5.5 presents the sub-classes of the main Location class and associated properties:

Table 5.5: Definition of the system database.

Subclass	Attributes
Points of Interest	Price, timetable, availability, accessibility
Accommodations	Price, availability, accessibility
Nature	Timetable, accessibility

5.2.2 Initial Proof-of-Concept Assistant

After the definition of the ontology and the database, the next step was to create a system that could integrate both of them using the framework presented in 3. The system was iteratively tested, debugged and improved by:

- Initial graph design of the state machine;
- Conversion of the graph to the dialogue manager and ontology files;
- Integrating the ontology and database;
- Addition of states and state transitions that allow the process of more requests, giving the dialogue system more functionalities;
- Adding alternatives to how a request can be completed.

Figure 5.1 presents the initial designed graph using Graphviz (GraphViz, 1991). The figure contains the system and user initial states that allow the process of a simple request like

`"Estou à procura de museus"` (`I'm looking for museums"`).

Beyond defining system and user states, this initial graph defines what triggers the transition between said states, containing ontology elements, such as `museus` and `Lisboa`.

Using the script detailed in section 3.1.5, this initial graph was transformed into the first iteration of the dialogue manager definition, as well as the ontology file, seen in figures 5.2 and 5.3.

With the initial ontology definition, the next step was to complete the ontology with elements that fit the scenario and the scope of the proof-of-concept (see section 5.2.1). Due to time restrictions, the examples are restricted to Lisboa and Aveiro with few examples for each.

Finally, the dialogue manager was improved to be capable of processing requests regarding the information in the ontology. This implied the addition of new states and state transitions, making the assistant more robust and functional.

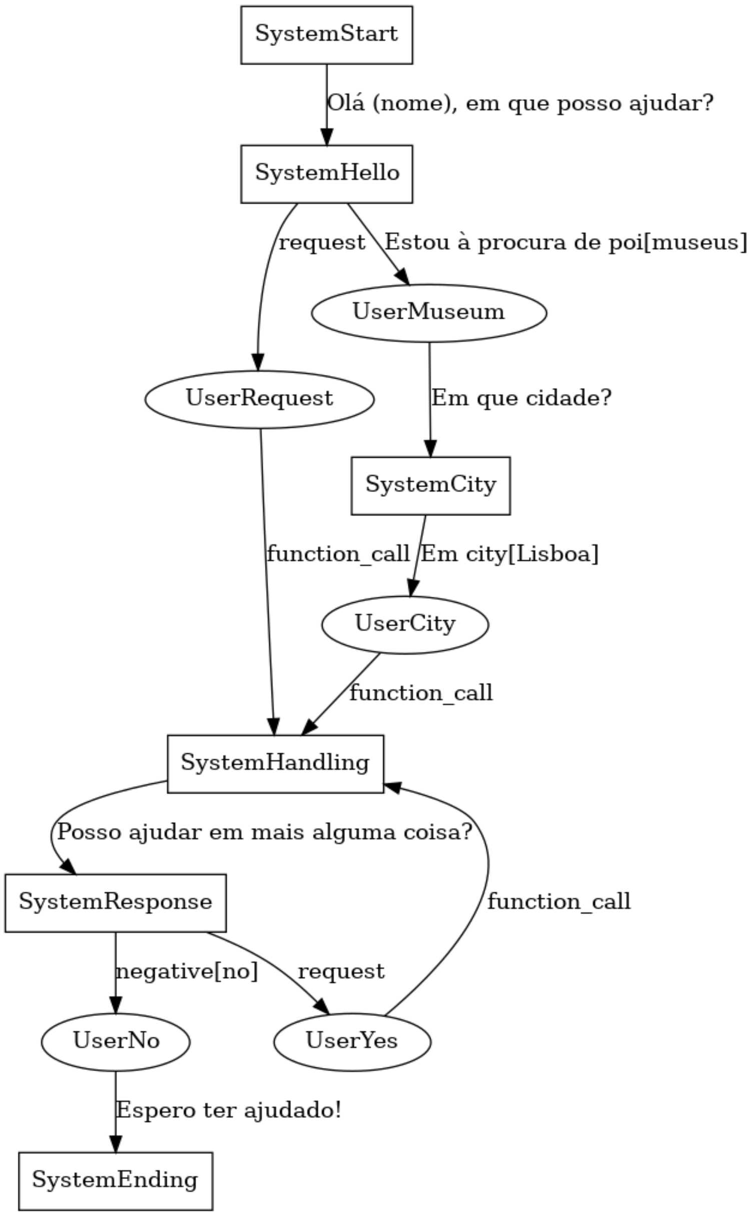


Figure 5.1: Initial tourism designed graph containing states, their transitions and ontology elements.

```
# AUTO GENERATED FILE
from emora_stdm import DialogueFlow, Macro, KnowledgeBase, NatexNLG
from enum import Enum, auto
import json

# WARNING: complete ontology file found at turismo.json
ontology = {}

with open('turismo.json') as json_file:
    ontology = json.load(json_file)

class State(Enum):
     # system states
     SystemStart = auto()
     SystemHello = auto()
     SystemEnding = auto()
     SystemResponse = auto()
     SystemHandling = auto()
     SystemCity = auto()
     # user states
     UserRequest = auto()
     UserYes = auto()
     UserNo = auto()
     UserCity = auto()
     UserMuseum = auto()

knowledge = KnowledgeBase()
knowledge.load_json(ontology)
df = DialogueFlow(State.SystemStart, initial_speaker =
DialogueFlow.Speaker.SYSTEM, kb = knowledge)

df.add_system_transition(State.SystemStart, State.SystemHello,
'"Olá (nome), em que posso ajudar?"')
df.add_system_transition(State.UserMuseum, State.SystemCity, '"Em que cidade?"')
df.add_system_transition(State.UserCity, State.SystemHandling, '"function_call"')
df.add_system_transition(State.UserRequest, State.SystemHandling,
'"function_call"')
df.add_system_transition(State.SystemHandling, State.SystemResponse,
'"Posso ajudar em mais alguma coisa?"')
df.add_system_transition(State.UserYes, State.SystemHandling, '"function_call"')
df.add_system_transition(State.UserNo, State.SystemEnding, '"Espero ter ajudado!"')
df.add_system_transition(State.SystemHello, State.UserRequest, '"request"')
df.add_system_transition(State.SystemHello, State.UserMuseum, '"Estou à procura
de poi[museus]"')
df.add_system_transition(State.SystemCity, State.UserCity, '"Em city[Lisboa]"')
df.add_system_transition(State.SystemResponse, State.UserYes, '"request"')
df.add_system_transition(State.SystemResponse, State.UserNo, '"negative[no]"')
```

Figure 5.2: The generated dialogue manager definition using the script presented in section 3.1.5 that transforms a textual representation of a graph into the definition of the Emora+ dialogue manager.

```
{
    "ontology": {
        "poi": [
            "museus"
        ],
         "city": [
             "Lisboa"
         ],
         "negative": [
             "no"
         ]
    }
}
```

Figure 5.3: The generated initial ontology definition.

5.3 Examples of Usage

Since this proof-of-concept is an initial approach, no real users tested the system but the author. Figures 5.4 and 5.5 present two examples of interactions with the assistant.

In the first example, the user was looking for an hotel in Aveiro with medium pricing. The assistant is capable of recognizing all the information from the user, providing him the answer he was looking for.

```
SYSTEM > Olá André , em que posso ajudar?
USER > estou à procura de um hotel em aveiro de preço medio
SYSTEM > Em Aveiro, o hotel Melia oferece preços nessa gama
```

Figure 5.4: First example of interaction with the assistant. The user is looking for a medium priced hotel in Aveiro.

In the second example, the user was looking for a park to visit. The assistant processes the input and asks the user the missing information to complete the request.

```
SYSTEM > Olá André , em que posso ajudar?
USER > estou à procura de um parque para visitar
SYSTEM > Está à procura de parques em que cidade?
USER > em lisboa
SYSTEM > Em Lisboa, tem vários parques muito conhecidos como o Parque
Eduardo VII, o Jardim da Estrela e o Jardim Mário Soares.
```

Figure 5.5: Second example of interaction with the assistant. The user wants to visit a park in Lisbon.

In the final example, the user is looking for a place to visit with good accessibility.

```
SYSTEM > Olá André , em que posso ajudar?
USER > estou à procura de um sítio para visitar
SYSTEM > Em que cidade?
USER > em lisboa
SYSTEM > Em Lisboa, pode visitar o Oceanário.
USER > tem acessibilidade para cadeira de rodas?
SYSTEM > Sim, o Oceanário tem boa acessibilidade.
```

Figure 5.6: Third example of interaction with the assistant. The user wants to visit a point of interest with good accessibility in Lisbon.

CHAPTER 6

Conclusions

This final chapter presents a summary of the work done by the author, highlighting the main results achieved.

In the end are also presented some suggestions for future work to be done, concluding this document.

6.1 Work Summary

The work was developed in several phases, namely:

Knowledge Acquisition: The first and one of the most important tasks for this work was gathering the necessary information for it to arise.

The study of relevant references, with special emphasis to (Jurafsky and Martin, 2009b), served as the basis to everything that came after. Studying relevant related research allowed the author to perceive communication, whether between humans or human and machine, in a whole different level. The characterization of the communication and dialogue itself, as well as understanding the different components that compose such systems, helped understanding how they work and how they are able to interact with humans.

Search and Testing of Tools: This was the longest task of all the work, since several different frameworks were extensively tested, and only a few were able to comply with requirements.

Selection and Decision: Following the study and testing with different tools came the selection of which to use as basis for this work. The selection of the framework to use on this work was one of the most crucial steps to take, as it had to be able to do specific tasks. First, they had to allow the development of systems capable of processing Portuguese inputs and generating outputs in the same language. Then, they had to be as much as possible natural with few and not hard-coded data. They also had to comply with these two requirement while maintaining a fully modular architecture. As a result, Emora was the chosen framework to handle the dialogue management and Plato to handle the understanding of the user inputs.

Improvements to the Selected Framework: When testing Emora, some adjustments felt needed. So, several improvements were made. Furthermore, it was also at this step that there was the integration of Plato with Emora to handle the task of natural language understanding.

Dialogue System for Smart Home: With the improved framework, a first proof-of-concept Assistant was designed and developed, aligned with the Smart Green Homes [1] project. It implied the definition of scenarios and requirements that later supported the definition of the system ontology. A requirement for this system was the inclusion of the back-end system developed in (Ketsmur et al., 2019), as part of the Smart Green Homes project.

The assistant developed in this scope was tested and evaluated by a small group of people.

Dialogue System for Accessible Tourism: The second proof-of-work was developed with the same chosen frameworks after all the changes it went through. This was also an assistant developed aligned with a pre-existent project, the project ACTION [2]. This assistant was developed aimed to users with accessibility needs, e.g., impaired movement or vision. The development steps were the same as the previous concept.

6.2 Main Results

Several relevant results were obtained, worth being highlighted the following:

- The first result is the in-depth knowledge obtained regarding spoken dialogue systems: their capacities, limitations and different existent strategies to develop them;

- The second and main result of this work is the developed Emora+ framework. It results from a combination of improvements made to Emora (Finch and Choi, 2020) with the integration of Plato (Papangelis et al., 2020).

- This improved development framework made possible two other important results: an assistant for a Smart Home demonstrator (a T0 implemented on the UX space at Bosch Thermotechnology in Aveiro) that was put to some initial user tests; and an initial version of an assistant for accessible tourism.

[1] `https://www.ua.pt/pt/smartgreenhomes/`
[2] `http://action.web.ua.pt/`

6.3 Future Work

There are several common points that can be improved on both of the developed proof-of-concepts. Foremost, the spoken dialogue system for accessible tourism can have its features evolved with real data. Regarding the assistant for Smart Homes, the improvement of on-boarding would facilitate the user task. An extension to the already developed features is also something that can be explored with devices and requests that make sense to the scope.

The generation of natural language as well as the understanding can be improved with machine learning techniques as long as there is enough data to train neural networks.

Finally, but also important, would be the testing of the smart home assistant in an ambient close to the real use scenario, planned for this work but made impossible by the present Pandemic situation.

References

Abbeduto, Leonard (1983). “Linguistic communication and speech acts. Kent Bach &amp; Robert M. Harnish. Cambridge: MIT Press, 1979, Pp. xvii 327.” In: *Applied Psycholinguistics* 4.4, 397–407. DOI: 10.1017/S0142716400004768.

Allen, James (1995). *Natural language understanding.* Pearson.

Amazon (2014). *Amazon Alexa.* https://www.alexa.com/. Online; accessed 4 november 2020.

Andrade, Guilherme, Geovana Silva, Francisco Júnior, Giovanni Santos, Fábio Lucio Mendonça, and Rafael de Sousa Junior (Jan. 2020). “EvaTalk: A Chatbot System for the Brazilian Government Virtual School”. In: *ICEIS (1)*, pp. 556–562. DOI: 10.5220/0009418605560562.

Apple (2011). *Apple Siri.* https://www.apple.com/siri/. Online; accessed 4 november 2020.

Arora, Suket, Kamaljeet Batra, and Sarabjit Singh (2013). *Dialogue System: A Brief Review.* arXiv: 1306.4134 [cs.CL].

Aust, Harald, Martin Oerder, Frank Seide, and Volker Steinbiss (1995). “The Philips automatic train timetable information system”. In: *Speech Communication* 17.3. Interactive Voice Technology for Telecommunication Applications, pp. 249 –262. DOI: https://doi.org/10.1016/0167-6393(95)00028-M. URL: http://www.sciencedirect.com/science/article/pii/016763939500028M.

Austin, J.L., J.L. Austin, J.O. Urmson, J.O. Urmson, and M. Sbisà (1975). *How to Do Things with Words.* A Harvard paperback. Harvard University Press. URL: https://books.google.pt/books?id=V43VS07TGEMC.

Bocklisch, Tom, Joey Faulker, Nick Pawlowski, and Alan Nichol (Dec. 2017). “Rasa: Open Source Language Understanding and Dialogue Management”. In: *arXiv preprint arXiv:1712.05181.*

Bohus, Dan and Alexander I. Rudnicky (July 2009). “The RavenClaw Dialog Management Framework: Architecture and Systems”. In: *Comput. Speech Lang.* 23.3, 332–361.

Bratman, Michael E., David J. Israel, and Martha E. Pollack (1988). “Plans and resource-bounded practical reasoning”. In: *Computational Intelligence* 4.3, pp. 349–355. DOI: 10.1111/j.1467-8640.1988.tb00284.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8640.1988.tb00284.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.1988.tb00284.x.

Chu, Shiu-Wah, Ian O’Neill, Philip Hanna, and Michael Mctear (Jan. 2005). “An approach to multi-strategy dialogue management”. In: *Ninth European Conference on Speech Communication and Technology*, pp. 865–868.

Clark, Herbert H. (Mar. 1996). “Herbert H. Clark, Using language. Cambridge: Cambridge University Press, 1996. Pp. xi+432.” In: *Journal of Linguistics* 35, pp. 167 –222. DOI: 10.1017/S0022226798217361.

Clark, Herbert H. and Edward F. Schaefer (1989). “Contributing to Discourse”. In: *Cognitive Science* 13.2, pp. 259–294. DOI: 10.1207/s15516709cog1302_7. eprint: https://onlinelibrary.wiley.

com/doi/pdf/10.1207/s15516709cog1302_7. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1302_7.

Dutoit, Thierry (1997). *An introduction to text-to-speech synthesis.* Vol. 3. Springer Science & Business Media.

Ferguson, George, Jill Quinn, Cecilia Horwitz, M Swift, J Allen, and Lucian Galescu (2010). "Towards a personal health management assistant". In: *Journal of biomedical informatics* 43.5, S13–S16.

Finch, James D. and Jinho D. Choi (2020). *Emora STDM: A Versatile Framework for Innovative Dialogue System Development.* arXiv: 2006.06143 [cs.CL].

Gao, Xiang, Sungjin Lee, Yizhe Zhang, Chris Brockett, Michel Galley, Jianfeng Gao, and Bill Dolan (2019). *Jointly Optimizing Diversity and Relevance in Neural Response Generation.* arXiv: 1902.11205 [cs.CL].

Goodwin, Charles and John Heritage (1990). "Conversation Analysis". In: *Annual Review of Anthropology* 19.1, pp. 283–307. DOI: 10.1146/annurev.an.19.100190.001435. eprint: https://doi.org/10.1146/annurev.an.19.100190.001435. URL: https://doi.org/10.1146/annurev.an.19.100190.001435.

GraphViz (1991). *Graph Visualization Software.* https://graphviz.org/. Online; accessed 13 november 2020.

Grice, Herbert P (1975). "Logic and conversation". In: *Speech acts.* Brill, pp. 41–58.

Grosch, Josef (2002). "Lark-An LALR(2) parser generator with backtracking". In:

Hutchby, Ian and Robin Wooffitt (July 1998). "Ian Hutchby & Robin Wooffitt, Conversation analysis: Principles, practices and applications". In: *Language in Society - LANG SOC* 29, pp. 416–419. DOI: 10.1017/S0047404500233046.

Issar, S. and W. Ward (1993). "CMLPs robust spoken language understanding system". In: *EUROSPEECH.*

Jurafsky, Daniel and James H. Martin (2009b). *Speech and Language Processing (2nd Edition).* USA: Prentice-Hall, Inc.

— (2009a). "Dialogue and Conversational Agents". In: *Speech and Language Processing (2nd Edition).* Ed. by Daniel Jurafsky and James H. Martin. Vol. 2. USA: Prentice-Hall, Inc. Chap. 24.

Ketsmur, Maksym, António Teixeira, Nuno Almeida, and Samuel Silva (July 2019). "Towards European Portuguese Conversational Assistants for Smart Homes". In: p. 14. DOI: 10.4230/OASIcs.SLATE.2019.5.

Ketsmur, Maksym, António Teixeira, Nuno Almeida, Samuel Silva, and Mário Rodrigues (June 2018). "Conversational Assistant for an Accessible Smart Home: Proof-of-Concept for Portuguese". In: DOI: 10.1145/3218585.3218594.

Larsson, Staffan and David R. Traum (Sept. 2000). "Information State and Dialogue Management in the TRINDI Dialogue Move Engine Toolkit". In: *Nat. Lang. Eng.* 6.3–4, 323–340. DOI: 10.1017/S1351324900002539. URL: https://doi.org/10.1017/S1351324900002539.

Levin, E., R. Pieraccini, and W. Eckert (2000). "A stochastic model of human-machine interaction for learning dialog strategies". In: *IEEE Transactions on Speech and Audio Processing* 8.1, pp. 11–23. DOI: 10.1109/89.817450.

Levinson, Stephen C. (1983). *Pragmatics.* Cambridge Textbooks in Linguistics. Cambridge University Press. DOI: 10.1017/CBO9780511813313.

Li, Jiwei, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan (2016). *A Diversity-Promoting Objective Function for Neural Conversation Models.* arXiv: 1510.03055 [cs.CL].

Lison, Pierre and Casey Kennington (Jan. 2016). “OpenDial: A Toolkit for Developing Spoken Dialogue Systems with Probabilistic Rules”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Demonstrations)*, pp. 67–72. DOI: 10.18653/v1/P16-4012.

Liu, Xiaodong, Yelong Shen, Kevin Duh, and Jianfeng Gao (2018). *Stochastic Answer Networks for Machine Reading Comprehension.* arXiv: 1712.03556 [cs.CL].

Ljunglöf, Peter (2009). “TrindiKit.py: An open-source Python library for developing ISU-based dialogue systems”. In: *Proc. of IWSDS* 9.

Luan, Yi, Chris Brockett, Bill Dolan, Jianfeng Gao, and Michel Galley (2017). *Multi-Task Learning for Speaker-Role Adaptation in Neural Conversation Models.* arXiv: 1710.07388 [cs.CL].

McTear, Michael F (2004). *Spoken dialogue technology: toward the conversational user interface.* Springer Science & Business Media.

Microsoft (2014). *Microsoft Cortana.* https://www.microsoft.com/en-us/cortana. Online; accessed 3 november 2020.

Norman, Donald A. (2002). *The Design of Everyday Things.* USA: Basic Books, Inc.

Papangelis, Alexandros, Mahdi Namazifar, Chandra Khatri, Yi-Chia Wang, Piero Molino, and Gokhan Tur (2020). *Plato Dialogue System: A Flexible Conversational AI Research Platform.* arXiv: 2001.06463 [cs.HC].

Pomerantz, Anita (1985). “Agreeing and disagreeing with assessments: some features of preferred/dispreferred turn shapes”. In: *Structures of Social Action.* Studies in Emotion and Social Interaction. Cambridge University Press, 57–101. DOI: 10.1017/CBO9780511665868.008.

Qin, Lianhui, Michel Galley, Chris Brockett, Xiaodong Liu, Xiang Gao, Bill Dolan, Yejin Choi, and Jianfeng Gao (2019). *Conversing by Reading: Contentful Neural Conversation with On-demand Machine Reading.* arXiv: 1906.02738 [cs.CL].

Sacks, Harvey, Emanuel A. Schegloff, and Gail Jefferson (1978). “A Simplest Systematics for the Organization of Turn Taking for Conversation”. In: *Studies in the Organization of Conversational Interaction.* Ed. by Jim Schenkein. Academic Press. Chap. 1, pp. 7 –55. DOI: https://doi.org/10.1016/B978-0-12-623550-0.50008-2. URL: http://www.sciencedirect.com/science/article/pii/B9780126235500500082.

Schegloff, Emanuel A. and Harvey Sacks (1973). “Opening up Closings”. In: *Semiotica* 8.4, pp. 289 –327. DOI: https://doi.org/10.1515/semi.1973.8.4.289. URL: https://www.degruyter.com/view/journals/semi/8/4/article-p289.xml.

Searle, John R. (1975). “A taxonomy of illocutionary acts”. In: *Expression and Meaning: Studies in the Theory of Speech Acts.* University of Minnesota Press, Minneapolis, 1–29. DOI: 10.1017/CBO9780511609213.003.

Shiv, Vighnesh Leonardo, Chris Quirk, Anshuman Suri, Xiang Gao, Khuram Shahid, Nithya Govindarajan, Yizhe Zhang, Jianfeng Gao, Michel Galley, Chris Brockett, Tulasi Menon, and Bill Dolan (July 2019). “Microsoft ICECAPS: An Open-Source Toolkit for Conversation Modeling”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations.* Florence, Italy: Association for Computational Linguistics, pp. 123–128. DOI: 10.18653/v1/P19-3021. URL: https://www.aclweb.org/anthology/P19-3021.

Singh, S., D. Litman, M. Kearns, and M. Walker (Feb. 2002). “Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System”. In: *Journal of Artificial Intelligence Research* 16, 105–133. DOI: 10.1613/jair.859. URL: http://dx.doi.org/10.1613/jair.859.

Stalnaker, Robert (2002). “Common ground”. In: *Linguistics and philosophy* 25.5/6, pp. 701–721.

Streebo (2018). *Chatbot for Airlines*. `https://www.streebo.com/chatbot-for-airlines`. Online; accessed 9 december 2020.

Traum, David R. and James F. Allen (1994). “Discourse Obligations in Dialogue Processing”. In: *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*. ACL ’94. Las Cruces, New Mexico: Association for Computational Linguistics, 1–8. DOI: `10.3115/981732.981733`. URL: `https://doi.org/10.3115/981732.981733`.

Traum, David R and Staffan Larsson (2003). “The information state approach to dialogue management”. In: *Current and new directions in discourse and dialogue*. Springer, pp. 325–353.

Ultes, Stefan, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić, and Steve Young (July 2017). “PyDial: A Multi-domain Statistical Dialogue System Toolkit”. In: *Proceedings of ACL 2017, System Demonstrations*. Vancouver, Canada: Association for Computational Linguistics, pp. 73–78. URL: `https://www.aclweb.org/anthology/P17-4013`.

Ward, Wayne and Sunil Issar (Jan. 1994). “Recent Improvements in the CMU Spoken Language Understanding System.” In: DOI: `10.3115/1075812.1075857`.

Weizenbaum, Joseph (Jan. 1966). “ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine”. In: *Commun. ACM* 9.1, 36–45. DOI: `10.1145/365153.365168`. URL: `https://doi.org/10.1145/365153.365168`.

Young, Steve (2007). *CUED standard dialogue acts*. Report, Cambridge University Engineering Department, 14th October. Cambridge University. URL: `https://rasa.com/`.