

Chatbot para serviços bancários

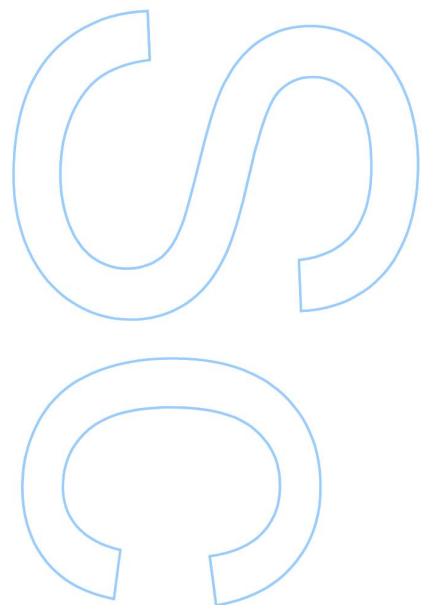
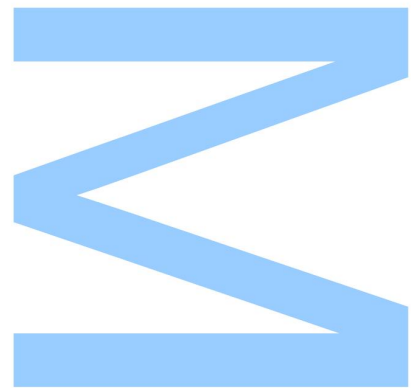
Fábio André Alves Teixeira

Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciência de Computadores
2018

Orientador

Alípio Mário Guedes Jorge, Professor Associado, Faculdade de Ciências da
Universidade do Porto



Abstract

Nowadays, people already want to stop going to the supermarket to do their shopping, since they buy most of their products online, such as clothes, shoes, gadgets, etc. There is, therefore, a clear need to avoid work and extra time that can be reused in another way. In this sense, chatbots arise. These programs, which try to simulate human conversations, are completely revolutionizing the methodology of business operation, ensuring a closer interaction with users and allowing the simpler delivery of their products.

In the banking area, the scenario is no different. If the application of ATMs was already revolutionary, the possibility of doing the same actions on a smartphone or computer takes the paradigm to an easier level of adhesion for the customer. Imagine now having the best of both worlds: the independent hypothesis of carrying out the planned actions 24/7, since there is no obligation to respect the working hours of the company's employees and at the same time to have a total and specialized service, which allows the customer greater satisfaction.

In this dissertation, a chatbot in Python was developed to respond to various banking services, from creating an account to making payments, allowing simple and direct interaction with the user through a series of generalization conversations. Three different approaches were used, the first through the Python Chatterbot library, in conjunction with a dialog dataset; the second by the manual creation of intents and use of NLTK tools; and finally the third one focused on an API for classification (LUIS, from Microsoft). In the first two approaches the results were not intended, since the intelligence of the bot was not guaranteed, which had no way of discovering the intention of the lines and could only handle the cases for which it was prepared. On the other hand, because of the possibility of discovering the intention of the utterances and the key values (entities) of the same, the use of the LUIS API proved to be highly efficient and the chosen course, ending by obtaining a work with a clear added value, focusing on the creation of intents and entities, program flow, code files, databases and even pseudo-code functions.

Resumo

Nos dias de hoje, as pessoas já querem deixar de se deslocar ao supermercado para fazer as suas compras, como também adquirem grande parte dos seus produtos via *online*, tais como roupa, sapatos, *gadgets*, por aí fora. Há, portanto, uma clara necessidade de evitar trabalho e tempo extra que podem ser reaproveitados de outra forma. Nesse sentido, surgem os *chatbots* – programas que tentam simular conversa humana –, que estão a revolucionar por completo a metodologia de funcionamento das empresas, garantindo uma interação mais próxima com os utilizadores e permitindo uma maior simplicidade na oferta dos seus produtos.

Na área bancária o cenário não é diferente. Se a aplicação de ATMs já foi revolucionária, a possibilidade de efetuar as mesmas ações sob um *smartphone* ou computador levou o paradigma para um nível de adesão mais facilitado para o cliente. Imagine-se, agora, ter o melhor de dois mundos: a hipótese independente de realizar as ações pretendidas 24/7 já que não há a obrigação de respeitar horários de trabalhos dos funcionários das empresas, e ao mesmo tempo ter um acompanhamento total, integral e especializado, que permite ao cliente uma maior satisfação pelo serviço.

Nesta Dissertação, foi desenvolvido um *chatbot* em Python que responde a diversos serviços bancários, desde a criação de uma conta até à realização de transferências ou pagamentos, permitindo uma interação simples e direta com o utilizador, através de uma série de conversas de generalização. Foram utilizadas três abordagens diferentes, a primeira através da biblioteca Chatterbot do Python, em conjunto com um *dataset* de diálogos; a segunda pela criação manual de *intents* e uso de ferramentas do NLTK; e finalmente a terceira centrou-se na utilização de uma API para classificação (LUIS, da Microsoft). Nas duas primeiras abordagens os resultados não foram os pretendidos, uma vez que não era garantida a inteligência do *bot*, que não tinha forma de descobrir a intenção das falas e só conseguia tratar os casos para os quais estava preparado. Por outro lado, pela possibilidade de descobrir a intenção das falas e ainda os valores-chaves (entidades) da mesma, a utilização da API LUIS revelou-se altamente eficiente e o rumo escolhido, acabando, no final, por se obter um trabalho com um claro valor acrescentado, com foco na criação de *intents* e *entities*, no *flow* do programa, nos ficheiros de código, nas bases de dados e até nas funções em pseudo-código, podendo ser replicado pelos grandes bancos nacionais.

Agradecimentos

Para que este projeto fosse bem sucedido, contei com o apoio e incentivo de diversas pessoas e entidades, pelo que tenho de agradecer:

Ao Prof. Alípio Jorge, pela forma como me orientou, atenção, disponibilidade, sensatez e valor das suas opiniões;

À Cláudia Almeida, pelo apoio, ajuda e carácter, que sempre me incentivaram a concluir o trabalho;

À minha família, pelas condições que me prestaram para que pudesse, dia após dia, trabalhar sem interrupções;

Ao DCC, FCUP e respetivos docentes e funcionários, pela constante aprendizagem e evolução tanto a nível de conteúdos escolares como pessoais nos últimos 5 anos.

Obrigado a todos.

Dedico à minha família e namorada

Conteúdo

Abstract	i
Resumo	ii
Agradecimentos	iii
Conteúdo	vii
Lista de Tabelas	viii
Lista de Figuras	x
Lista de Blocos de Código	xi
Acrónimos	xii
1 Introdução	1
2 Enquadramento técnico	5
3 Background	6
4 Estado da Arte	10
4.1 Componentes de um <i>chatbot</i>	10
4.2 Estado da investigação	11
4.2.1 Interfaces de conversação	13
4.2.2 Casos de utilização	14

4.3	Produtos comerciais	19
4.3.1	Plataformas para desenvolvimento de <i>bots</i>	19
4.3.2	<i>Frameworks</i> para desenvolvimento de <i>bots</i>	19
4.3.3	AIML	20
4.3.4	NLTK	20
4.4	Biblioteca Chatterbot do Python	21
5	Motivação para o desenvolvimento	24
5.1	Chatterbot	24
5.2	Criação de <i>intents</i> + NLTK	26
6	Desenho e Desenvolvimento	29
6.1	LUIS	29
6.2	Integração com o Azure	30
6.3	Código e Linguagem	31
6.4	Diagrama de ficheiros	31
6.5	<i>Flow</i> do Programa	34
6.6	Base de dados	36
6.6.1	Criação	37
6.6.2	<i>Update</i>	37
6.6.3	Operações	38
7	Experiências e Testes	40
7.1	LUIS vs. Watson	40
7.2	Testes quantitativos	41
8	Resultados e análise	46
8.1	Valor acrescentado	46
8.2	Colaboração com a documentação do Chatterbot	46
8.3	Ficheiros de texto com diálogos:	47

8.4	Funções em pseudo-código	47
8.5	Repositório no GitHub	47
9	Conclusões	50
9.1	Trabalho Futuro	51
	Bibliografia	52

Lista de Tabelas

6.1	<i>Intents</i> seguintes.	35
6.2	Exemplo de respostas para os <i>intents</i> de <i>small talk</i>	36

Lista de Figuras

1.1	Exemplo de um diálogo bancário com o <i>chatbot</i> proposto.	3
1.2	Entradas criadas na base de dados depois da conversa.	4
3.1	Interação com o ELIZA. Retirado de https://en.wikipedia.org/wiki/ELIZA	7
3.2	Diferenciação entre a utilização de aplicações de mensagens e redes sociais de 2011 a 2015. Fonte: <i>Companies, BI Intelligence</i>	8
4.1	Componentes de um <i>chatbot</i>	11
4.2	Representação esquemática do processo de extração do contexto num <i>chatbot</i> [1].	14
4.3	WellsFargoBankingAssistant.	16
4.4	HSBC Virtual Assistant.	17
4.5	Os 5 maiores bancos dos EUA que adotaram <i>chatbots</i> . Retirado de Maruti Techlabs.	18
4.6	O <i>chatbot</i> A.L.I.C.E., programado em 1950, através da linguagem AIML.	20
4.7	Exemplo de utilização dos <i>Tokenizers</i> no NLTK.	21
4.8	<i>Process Flow Diagram</i> da biblioteca do Chatterbot.	23
5.1	Ficheiros do <i>dataset</i> de diálogos.	25
5.2	Um exemplo de conversação e interação com o <i>bot</i> . <i>Print screen</i> tirado a 16/01/2018.	26
6.1	<i>Intents</i> criados no LUIS.	30
6.2	<i>Entities</i> criadas no LUIS.	30
6.3	Exemplo de frases criadas no LUIS para o <i>intent</i> TransferirDinheiro e respetiva classificação de <i>entities</i>	31
6.4	Ligações entre os ficheiros.	33

6.5	Arquitetura do <i>chatbot</i> proposto.	36
6.6	Exemplo de entrada na base de dados.	37
8.1	Ficheiros do <i>dataset</i> de diálogos.	46

Lista de Blocos de Código

5.1	Requerimentos para utilizar a biblioteca Chatterbot	24
5.2	Código do <i>bot</i> utilizando a biblioteca Chatterbot.	25
5.3	Exemplos de <i>intents</i> utilizados com o NLTK.	27
6.1	Classificação para a <i>query</i> exemplo.	32
6.2	Receber dados JSON no código.	33
6.3	Compilar <i>bot</i>	34
6.4	Excerto do <i>main</i>	34
6.5	Criar base de dados e tabelas.	37
6.6	Criar utilizador.	38
6.7	Criar conta.	38
6.8	Saber saldo.	39
6.9	Atualizar valores.	39
7.1	Classificação para a <i>query</i> exemplo.	41
7.2	Classificação para a <i>query</i> exemplo.	42
7.3	Classificação para a <i>query</i> exemplo.	42
7.4	Classificação para a <i>query</i> exemplo.	42
7.5	Classificação para a <i>query</i> exemplo.	43
7.6	Classificação para a <i>query</i> exemplo.	43
7.7	Classificação para a <i>query</i> exemplo.	43
7.8	Classificação para a <i>query</i> exemplo.	44
7.9	Classificação para a <i>query</i> exemplo.	44
7.10	Classificação para a <i>query</i> exemplo.	45
8.1	Pseudo-código (<i>main.py</i>).	47
8.2	Pseudo-código (<i>intents.py</i>).	48
8.3	Pseudo-código (<i>next_intent.py</i>).	49
8.4	Pseudo-código (<i>table.py</i>).	49
8.5	Pseudo-código (<i>ent.py</i>).	49
8.6	Pseudo-código (<i>s_talk.py</i>).	49

Acrónimos

AIML	Artificial Intelligence Markup Language	I/O	Input/Output
ATM	Automatic Teller Machine	IT	Information Technology
BD	Base de Dados	LUIS	Language Understanding Intelligent Service
DCC	Departamento de Ciência de Computadores	NLTK	Natural Language Toolkit
FCUP	Faculdade de Ciências da Universidade do Porto	NLU	Natural Language Understanding
IA	Inteligência Artificial	NLP	Natural Language Processing
IoT	Internet of Things	SaaS	Software as a Service

Capítulo 1

Introdução

Um *chatbot* é um *software* de inteligência artificial que pode simular uma conversa (semelhante a um *chat*) numa linguagem perceptível ao utilizador, como o português, o inglês, etc. A tecnologia, que cada vez reúne mais interesse, tem como grande propósito a otimização da interação entre humanos e máquinas, nomeadamente para impulsionar os negócios das empresas. Do ponto de vista tecnológico, um *chatbot* representa a evolução natural de um sistema de pergunta-resposta que utiliza *Natural Language Processing*. A formulação de respostas a perguntas em linguagem natural é um dos exemplos mais típicos de NLP aplicado a *software* que as empresas apresentam aos consumidores. Hoje em dia já é frequente encontrarmos os *bots* embebidos em aplicações de *messaging*, sites ou até *apps* móveis.

Ora, como estes *softwares* aumentam, em teoria, a rapidez com que as tarefas são realizadas e diminuem o trabalho que tais tarefas requerem e também a complexidade inerente, não é de estranhar que várias envolvências do mundo dos negócios estejam a apostar na tecnologia. Uma das áreas que se tem mostrado mais interessadas é a bancária, com diversos bancos a nível Internacional a criarem *bots* programáveis para aproximar a relação com o utilizador final, especialmente se tivermos em conta que é bastante mais fácil realizar os encargos quando há alguma conversa por trás do que quando não há interação de todo. Com isto torna-se possível realizar ações 24/7, já que não há a obrigação de respeitar horários de trabalhos dos funcionários das empresas, e ao mesmo tempo ter um acompanhamento total, integral e especializado, que permite ao cliente uma maior satisfação pelo serviço.

Os objetivos do trabalho centraram-se então em:

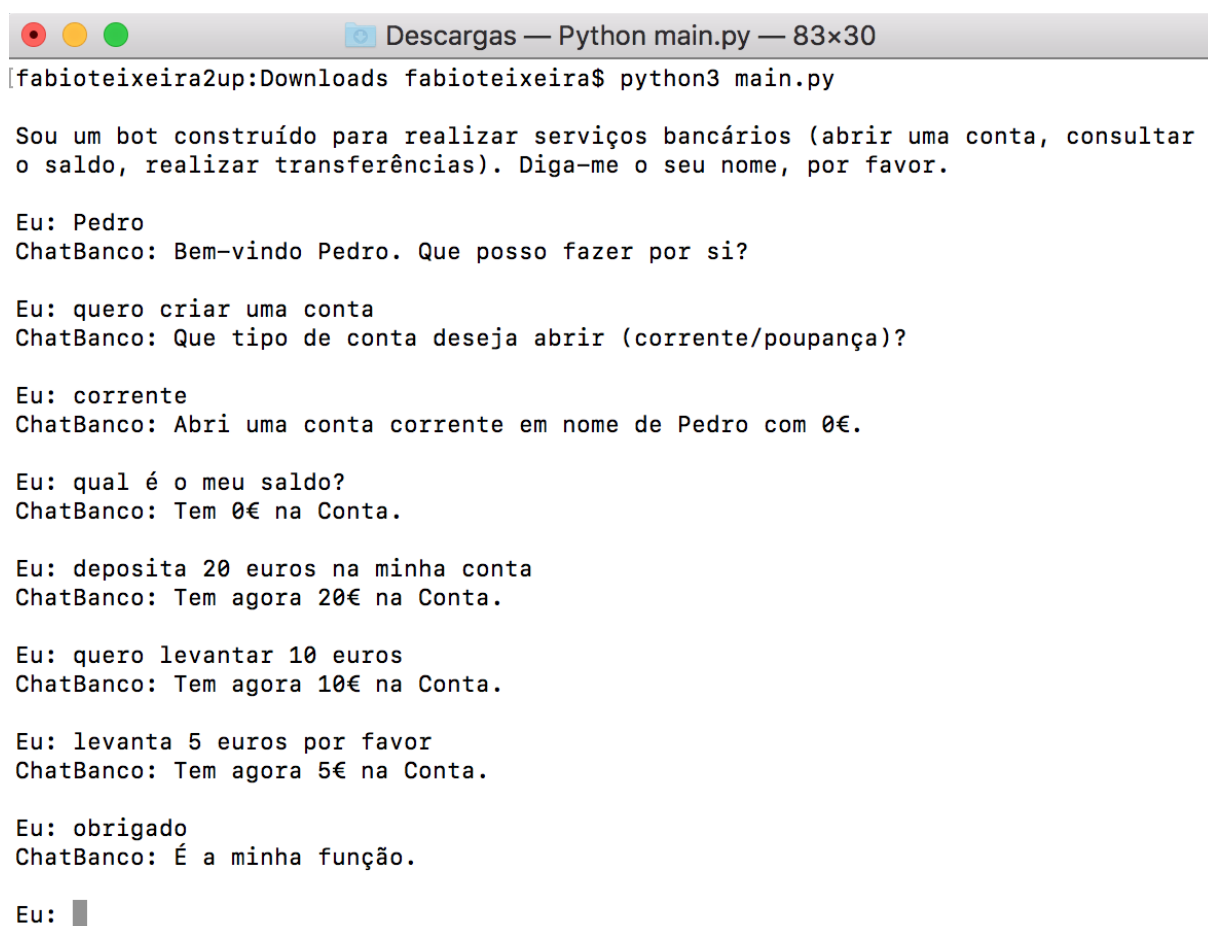
- Estudar a forma como os *chatbots* têm sido apresentados ao longo dos tempos;
- Analisar o estado corrente da tecnologia e o que tem sido feito na área;
- Procurar ferramentas existentes para possível adaptação futura;
- Produzir um protótipo demonstrável tão sofisticado quanto possível.

A abordagem seguida, depois de outras duas que se revelaram insuficientes, foi a utilização de uma API (*bot framework* LUIS da Microsoft) para classificação de intenções, que depois foi programada para transformar dados JSON num sistema de mensagens trocadas entre o utilizador e a máquina.

Entre os resultados obtidos encontram-se:

- Criação de um *bot* bancário em língua portuguesa;
- Intenções e entidades definidas no LUIS;
- Mais de 450 falas de diálogo;
- Ficheiros de texto com *intents*;
- *Flow* do programa com diagramas adjacentes;
- Definição dos *intents* seguintes;
- Respostas para cada *intent*;
- Funções definidas;
- Base de dados que simula a relação entre consumidor e banco;
- Mas acima de tudo um modelo inovador que pode ser replicado pelos grandes bancos nacionais, uma vez que ainda não há um *chatbot* desta estirpe em Portugal que trate dos serviços bancários dos seus utilizadores de forma independente (sem qualquer ajuda humana) e que ao mesmo tempo possibilite uma conversa em tempo real.

O presente relatório está organizado da seguinte forma: na secção 2 é apresentado o enquadramento técnico para a realização do trabalho; na secção seguinte (3) encontra-se retratado um pouco do *background* dos *chatbots*; posteriormente (secção 4) analisa-se o estado corrente da tecnologia, bem como ferramentas comerciais e produtos similares existentes; depois (secção 5) são exibidas as abordagens testadas antes do modelo final, mas que não foram escolhidas para o desenvolvimento; de seguida (secção 6) é detalhado todo o processo de criação do *bot*, em conjunto com blocos de código e diagramas auxiliares; há ainda espaço para experiências e testes (secção 7) e resultados obtidos (secção 8); e por fim faz-se uma revisão sintética da oportunidade que representam os *chatbots*, assim como o que poderá ser o futuro do trabalho (secção 9).



A terminal window titled "Descargas — Python main.py — 83x30" displays a conversation between a user and a chatbot named ChatBanco. The chatbot is designed to simulate a bank, handling requests to open accounts, check balances, deposit, and withdraw money. The user, Pedro, interacts with the bot by providing personal information and making transactions. The chatbot responds with appropriate banking-related messages, such as account creation, balance updates, and transaction confirmations. The conversation ends with the user saying "obrigado" (thank you) and the chatbot responding "É a minha função." (It's my function).

```
[fabioteixeira2up:Downloads fabioteixeira$ python3 main.py]

Sou um bot construído para realizar serviços bancários (abrir uma conta, consultar
o saldo, realizar transferências). Diga-me o seu nome, por favor.

Eu: Pedro
ChatBanco: Bem-vindo Pedro. Que posso fazer por si?

Eu: quero criar uma conta
ChatBanco: Que tipo de conta deseja abrir (corrente/poupança)?

Eu: corrente
ChatBanco: Abri uma conta corrente em nome de Pedro com 0€.

Eu: qual é o meu saldo?
ChatBanco: Tem 0€ na Conta.

Eu: deposita 20 euros na minha conta
ChatBanco: Tem agora 20€ na Conta.

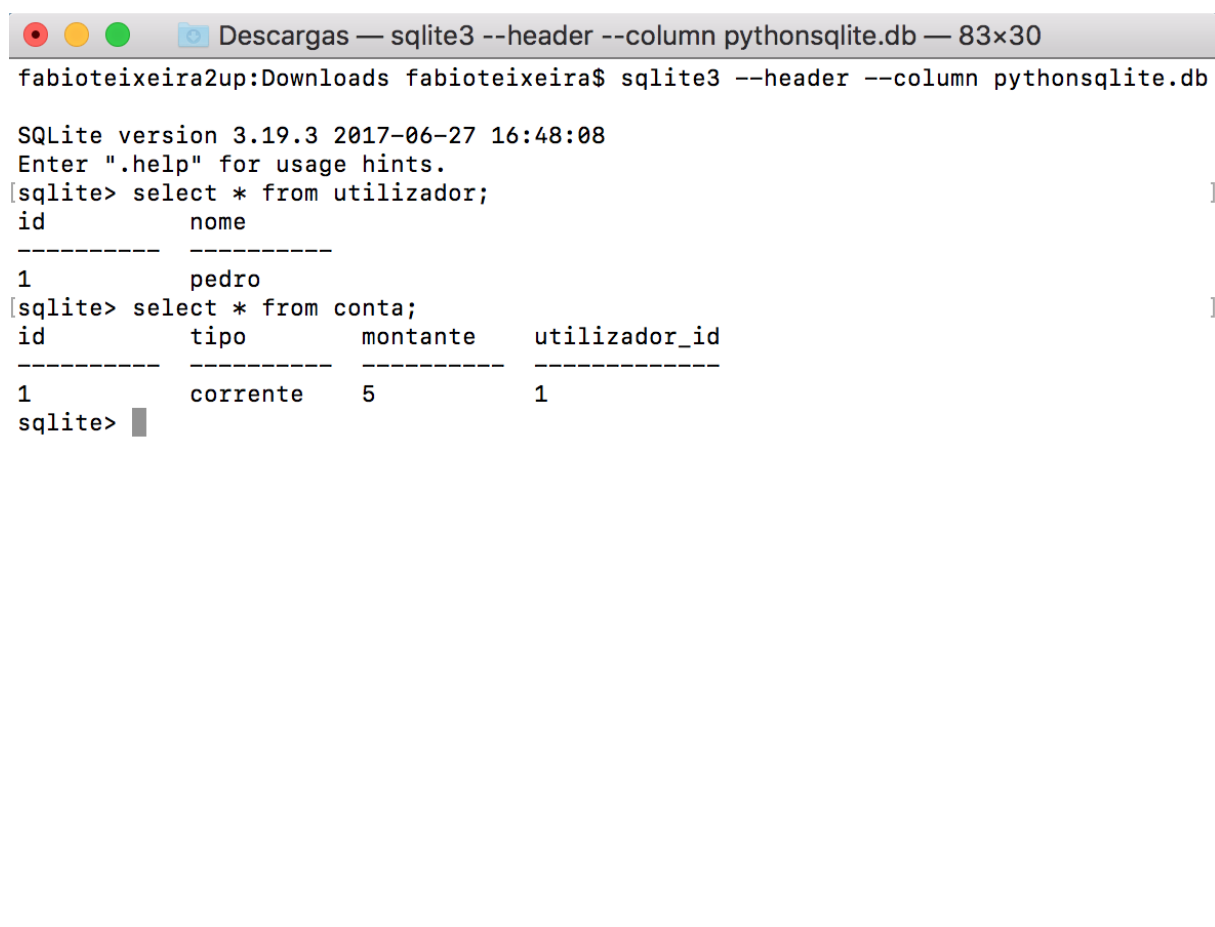
Eu: quero levantar 10 euros
ChatBanco: Tem agora 10€ na Conta.

Eu: levanta 5 euros por favor
ChatBanco: Tem agora 5€ na Conta.

Eu: obrigado
ChatBanco: É a minha função.

Eu: █
```

Figura 1.1: Exemplo de um diálogo bancário com o *chatbot* proposto.



A terminal window titled "Descargas — sqlite3 --header --column pythonsqlite.db — 83x30". The prompt is "fabioiteixeira2up:Downloads fabioiteixeira\$". The user runs "sqlite3 --header --column pythonsqlite.db". The terminal shows the SQLite version "3.19.3 2017-06-27 16:48:08" and a message "Enter '.help' for usage hints." The user enters "sqlite> select * from utilizador;". The output is a table with two columns: "id" and "nome". The first row has the value "1" for "id" and "pedro" for "nome". The user then enters "sqlite> select * from conta;". The output is a table with four columns: "id", "tipo", "montante", and "utilizador_id". The first row has the values "1", "corrente", "5", and "1". The prompt "sqlite>" is shown at the end of the output.

```
fabioiteixeira2up:Downloads fabioiteixeira$ sqlite3 --header --column pythonsqlite.db

SQLite version 3.19.3 2017-06-27 16:48:08
Enter ".help" for usage hints.
sqlite> select * from utilizador;
id      nome
-----
1       pedro
sqlite> select * from conta;
id      tipo      montante  utilizador_id
-----
1       corrente  5         1
sqlite>
```

Figura 1.2: Entradas criadas na base de dados depois da conversa.

Capítulo 2

Enquadramento técnico

O trabalho desenvolvido fez parte da Tese de Mestrado de Engenharia de Redes e Sistemas Informáticos e teve muito do que foi aprendido nas cadeiras de programação, de sistemas informáticos, de desenho e análise de algoritmos, de análise de *software*, de bases de dados, etc., que foram lecionadas ao longo do curso.

Para concluir o trabalho apenas foi necessário um computador com acesso à Internet, mas utilizou-se um *Bot Framework* da Microsoft chamado Azure, com uma chave de subscrição gratuita durante 30 dias e depois paga consoante a utilização de recursos.

A linguagem de programação escolhida para a realização da Dissertação não foi aprendida em nenhuma das disciplinas do curso, mas com a aprendizagem de outras linguagens como o #C ou o Java, ficou mais facilitada a sua interiorização.

As expressões que estão registadas nos acrónimos foram, também, todas elas embebidas nestes últimos 5 anos de Ensino Superior.

Capítulo 3

Background

Em 1950, a meio do século passado, Alan Turing¹ publicou o artigo *Computing Machinery and Intelligence*, no qual partilhou pela primeira vez o *Turing test*. Este teste permitia distinguir o comportamento de um programa de computador com um de um humano e acabou por se revelar altamente influenciador e um conceito fundamental na área da Inteligência Artificial[2].

A partir desta data os especialistas em *Computer Science* que queriam criar *software* inteligente começaram a ter sempre em conta o Teste de Turing, uma vez que ultrapassado o mesmo, havia, de facto, a garantia que o programa iria comportar-se como um simulador de consciência humana.

O professor Joseph Weizenbaum², em 1966, criou aquele que ficou conhecido como o primeiro *chatbot* da história, o ELIZA. Apesar do sucesso do *bot* e de parecer genuinamente inteligente, já que conversava como um humano, o produto de Weizenbaum não era mais do que um conjunto de respostas pré-programadas que eram emitidas sempre que alguma palavra-chave era referida e tinha apenas 204 linhas de código.

Os primeiros *bots* com inteligência apareceram apenas no século XXI e já envolviam Processamento de Linguagens Naturais[3], uma *feature* que se tornou indispensável para que estes reunissem condições de aprendizagem. O desenvolvimento da indústria ganhou, então, uma forte componente tecnológica, mas foi finalmente em 2016 que o estudo e construção dos *chatbots* sofreu um aumento de interesse brutal e tudo graças a uma mudança de paradigma que vinha a ser gradual, mas que poucos tinham reparado.

Em abril de 2016 o Facebook anunciou que o Messenger passaria a suportar conversações individuais com *bots* personalizados, o que exponenciou, de uma forma louca, o mediatismo destes simuladores. As empresas começaram a apostar na tecnologia, e ainda que tenham demorado algum tempo a afinar os programas (os primeiros *bots* foram um fracasso), os *chatbots* atingiram um nível de exuberância estratosférico.

¹Alan Mathison Turing (1912-1954) foi um matemático, lógico, criptoanalista e *computer scientist* britânico.

²Joseph Weizenbaum (1923-2008) foi um escritor e *computer scientist* teuto-americano. Professor emérito do MIT, também criou a linguagem de programação SLIP.

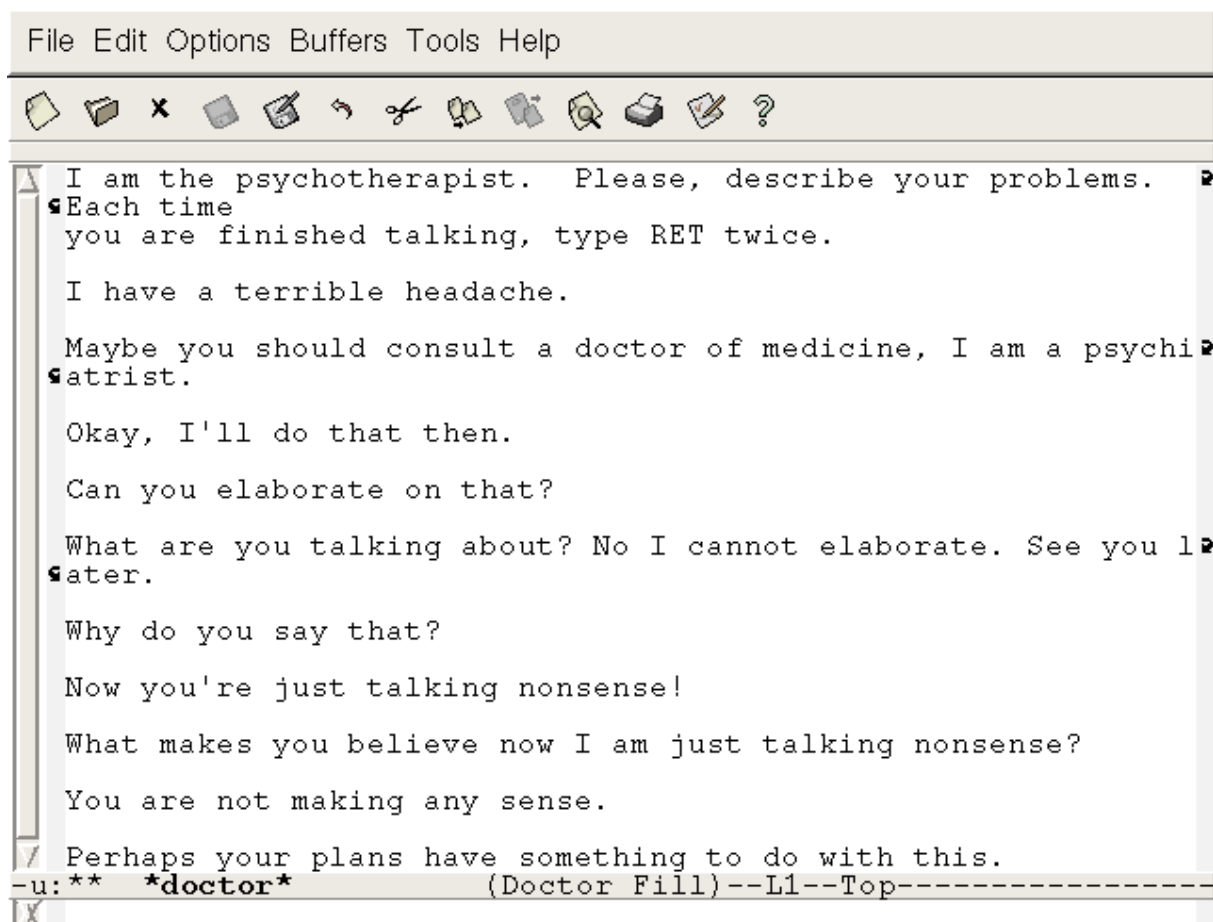


Figura 3.1: Interação com o ELIZA. Retirado de <https://en.wikipedia.org/wiki/ELIZA>.

“~90% do nosso tempo no telemóvel é passado em plataformas de e-mail e mensagens. Adoraria formar equipas que criassem coisas para sítios onde os consumidores vão!” - **Niko Bonatsos**, Diretor Geral na General Catalyst.

“As pessoas estão agora a passar mais tempo em aplicações de mensagens do que nas redes sociais, o que é um grande ponto de viragem. As aplicações de mensagens são as plataformas do futuro e os *bots* vão ser o meio para aceder a todos os tipos de serviço.” - **Peter Rojas**, empresário e trabalhador da Betaworks.

“As mensagens são os locais onde passamos uma grande parte do tempo e esperamos comunicar. É ridículo que ainda tenhamos que ligar para a maioria dos negócios.” - **Josh Elman**, sócio-parceiro na Greylock.

Ao trazer os *chatbots* para os locais onde as pessoas passavam mais tempo ao telemóvel, a adesão dos mesmos ganhou um enorme *boost*, transformando de uma forma gigantesca a forma como as empresas chegavam aos seus consumidores. Hoje em dia, outras plataformas como o Skype, o Slack ou o Telegram também já permitem o lançamento e implementação de *bots* nos seus serviços e é provável que mais ferramentas de *messaging* se juntem à lista.

Os *bots* estão em claro crescimento, mas, e como se estima que no futuro tenham um peso cada vez maior na vida das pessoas, este é um excelente momento para apostar na ideia, já que é mais fácil ter influência no mercado por diferenciação quando ainda não há sobrelotação. O facto de, como termos visto, as aplicações de *messaging* estarem a suplantiar as redes sociais como principal plataforma de utilização na Internet só vem exponenciar as vantagens de construir um *chatbot*.

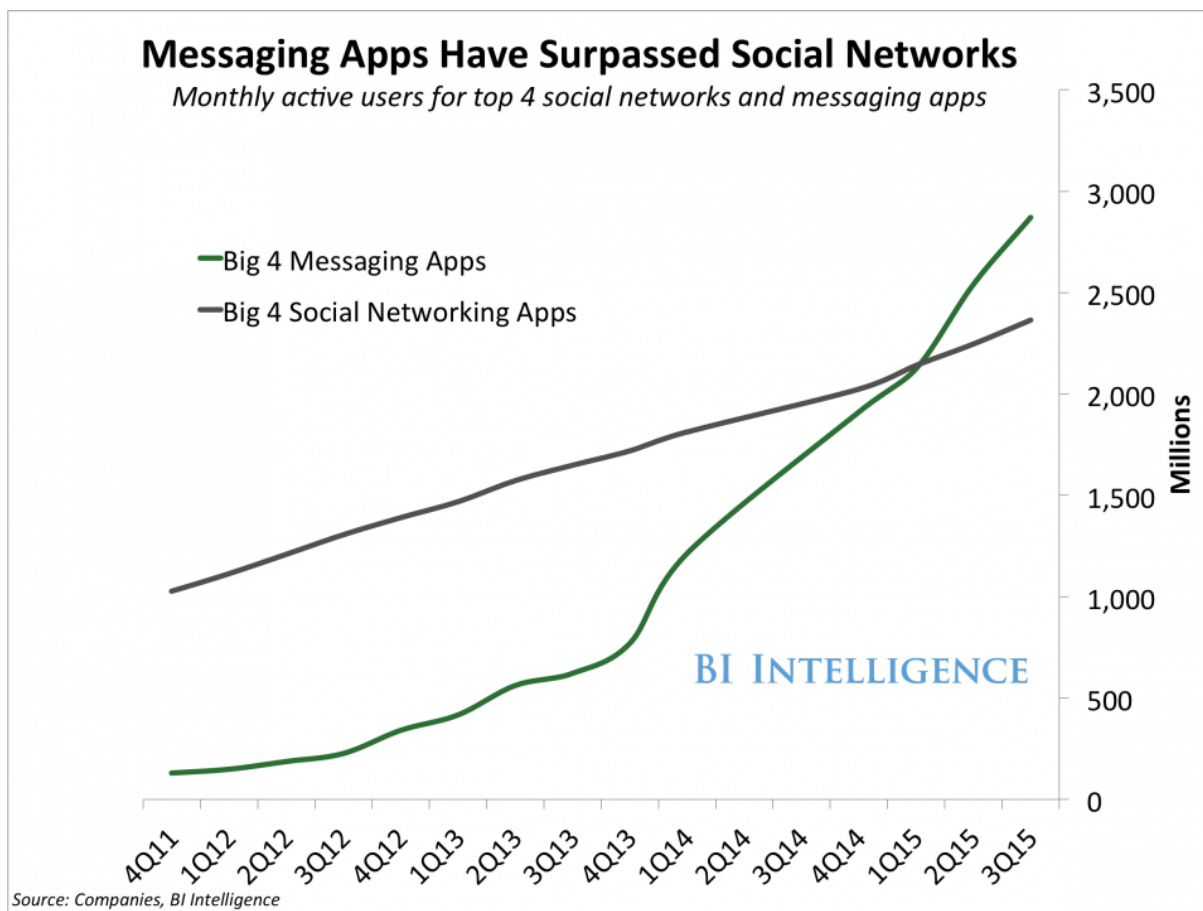


Figura 3.2: Diferenciação entre a utilização de aplicações de mensagens e redes sociais de 2011 a 2015. Fonte: *Companies, BI Intelligence*.

Há, no entanto, uma grande linha que pode ditar o sucesso de um *bot*, que é a sua capacidade de se aproximar do carácter humano, seja pela sua interação, linguagem, conselhos ou até simplicidade da interface. No setor da banca, as capacidades dos *bots* e as suas *features* permitem diferentes focos de inserção, desde apoio ao cliente até pagamentos e transferências, ou até transações de Blockchain. As aplicações dos *chatbots* podem estar embutidas nos *sites* e *apps* particulares dos bancos ou nas plataformas já existentes, como o Facebook Messenger, Skype, etc. Mas o grande desafio é fazer da versatilidade de opções dos Multibancos e ATMs - um dos sistemas embutidos mais revolucionários das últimas décadas - mais *frendly-user*, ou seja, menos complexas, mais rápidas e menos trabalhosas devido à não imposição de ter de realizar deslocamentos. Com isto, pretende-se um aumento na atividade dos clientes dos bancos, com

melhorias financeiras para as entidades e com uma maior interação e aproximação entre utilizador final e o fornecedor do serviço.

Capítulo 4

Estado da Arte

Os *chatbots* de hoje são desenhados para responder a propósitos muito específicos. Seja para vender produtos, fornecer uma linha de apoio ao cliente mais simplista, saber mais sobre concertos de uma banda, consultar o tempo, ler as notícias, etc. O objetivo é que forneçam um serviço mais rápido, conciso e que não obrigue as pessoas a longas pesquisas para obterem aquilo que pretendem.

Os *bots* dividem-se em duas categorias: os que detetam simplesmente *keywords* e respondem com frases pré-formuladas e os que ganham condições de aprendizagem com análises constantes das palavras trocadas ao longo do tempo, estudando a frequência com que aparecem, entre outros fatores, que os classificam como inteligência artificial.

As pequenas e grandes empresas começam a gerar *chatbots* para se aproximarem dos seus clientes e garantirem uma maior probabilidade de financiamento, por questões de acessibilidade e simplicidade, uma vez que é muito mais fácil adquirir produtos quando o processo para o fazer é minimalista de esforço.

Os *chatbots* têm, de facto, o potencial para substituir tarefas extensas e trabalhosas com um simples método de conversação e interação com sistemas designados a tais propósitos. Na área da banca, estima-se que, em 2020, os consumidores vão passar a gerir 85% de todos os seus negócios e associações com os bancos através de *chatbots* que vão funcionar e servir como assistentes financeiros pessoais[4].

4.1 Componentes de um *chatbot*

Para desenhar um *chatbot*, é necessário compreender e identificar as partes constituintes inerentes a estes pacotes de *software*. Assim, um *chatbot* pode ser dividido em 3 partes: *Responder*, *Classifier* e *Graphmaster*[5].

1. *Responder*

- É a parte que desempenha o papel da interface entre o *bot* e o utilizador. As tarefas do *Responder* incluem a transferência de dados do utilizador para o *Classifier* e o controlo do I/O.

2. *Classifier*

- É a parte entre o *Responder* e o *Graphmaster*. Esta camada filtra e normaliza o *input*, segmenta-o em componentes lógicos, transfere a frase normalizada para o *Graphmaster*, processa o *output* do *Graphmaster* e manipula as instruções de sintaxe da base de dados (por exemplo, AIML).

3. *Graphmaster*

- É a parte que executa as tarefas de organizar o conteúdo ou armazenar os algoritmos de correspondência de padrões.

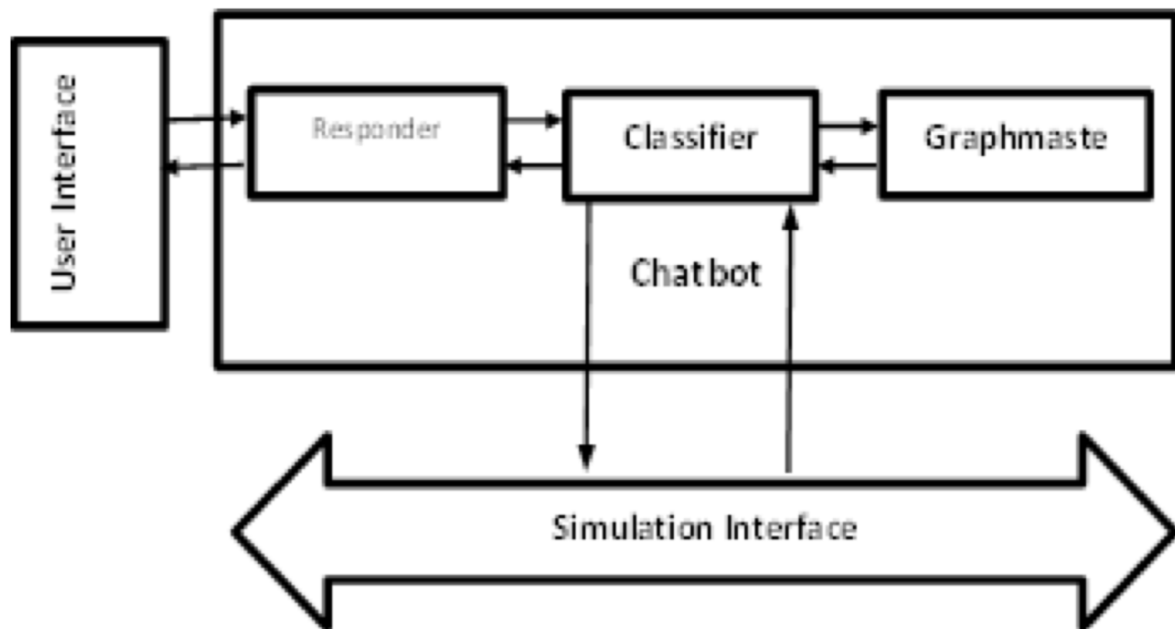


Figura 4.1: Componentes de um *chatbot*.

4.2 Estado da investigação

Os *chatbots* têm representado uma grande parte da investigação científica na área da tecnologia desde o início do século XXI, tendo ganho signicante importância na última década. Muitos autores têm desenvolvido projetos que incidem na implementação de *bots*, apesar da área bancária continuar a parecer mais um tema "para o futuro", por toda a dificuldade que representa construir um *chatbot* bancário.

No âmbito de um *chatbot*, o Processamento de Linguagens Naturais - que aborda a geração e compreensão automática de línguas humanas naturais - inclui 5 passos importantes[6]. A **análise léxica** corresponde à identificação e análise da estrutura das palavras e divide o texto por capítulos, frases e palavras. A **análise sintática** envolve a análise da gramática e organização das palavras, de modo a que as relações entre as palavras se tornem claras e façam sentido. A **análise semântica** verifica o texto para ver se tem significado e desenha esse exato significado mapeando construções sintáticas. **Integração do discurso** e **análise pragmática** trabalham na interpretação final da mensagem real do texto, já que o significado de qualquer frase depende do seu contexto.

Os sistemas de pergunta-resposta (*question answering*) têm fortes contornos de Processamento de Linguagens Naturais e podem ser divididos em duas categorias: domínio fechado[7] e domínio aberto[8]. O primeiro abrange apenas diálogo sob um específico tema, como compras de roupa ou meteorologia, e são naturalmente mais fáceis de codificar, obtendo também melhores resultados. O segundo concentra-se num sistema mais abrangente a outros temas, tendo de ser capaz de responder a perguntas que não estejam limitas apenas ao campo específico.

Os *bots* bancários estão, mais ou menos, no meio destas duas categorias. Por um lado não se podem comportar exclusivamente como um domínio fechado, pois têm de oferecer uma maior intenção do que aquela que é dada pelas ATMs em que só apenas só selecionadas as escolhas e permitir, de certa forma, ao utilizador que este se possa sentir confortável para prolongar o diálogo até um nível de confiança em que fique esclarecido e seguro com os *inputs* que está a receber. Por outro lado, os *chatbots* não podem ser tão díspares com possíveis diálogos, como outros que simulam um certo nível de conversa que se assemelha a um psicólogo[9]. Isto é, se queremos um *bot* que consiga falar normalmente e que não mostre desconhecimento para outras falas que receba que não sejam exclusivas da sua área de incisão, domínio fechado não funciona; se querermos tornar o *bot* eficiente e conduzir o utilizador para o seu âmbito de funcionamento, este também não se deve comportar inteiramente como um domínio aberto.

Além disto, os *bots* bancários representam ainda outra categoria dos sistemas de diálogo, que é o facto de serem orientado para tarefas. Assim, se um utilizador quer abrir uma conta e fornece os seus dados, o *chatbot* tem de estar associado a uma base de dados em que lhe seja possível fazer o *insert* dos dados do utilizador. Existem outras operações, nomeadamente a transferência de dinheiro, que já envolvem duas contas no banco e, por conseguinte, duas operações. E, em geral, os sistemas de diálogo orientados para tarefas fornecem uma estrutura para que quem utilize o *bot* tenha conversas com a máquina para concluir uma tarefa específica. Normalmente os *chatbots* são sistemas não orientados a tarefas, não tendo a intenção de ajudar o utilizador a concluir qualquer tarefa específica e só se concentrando nos diálogos, o que já é difícil de si[10], mas os *bots* bancários são ainda mais ardilosos de desenvolver.

Por todas estas questões, ainda não existe grande ênfase da comunidade científica dos *chatbots* na área bancária. Outra das condicionantes é a proteção e privacidade dos dados das pessoas, o que faz com que apenas os bancos possam ser autores de *bots*, uma vez que *developers* que queiram constuir os seus próprios *chatbots* realizar para serviços bancários, os tenham de fazer

sob um ambiente virtual como a criação de bases de dados, com representantes e contas fictícias.

4.2.1 Interfaces de conversação

Bots da área bancária podem funcionar apenas como *assistant systems*, que direcionam os utilizadores para outros serviços que - esses sim -, resolverão o problema[11]. Estes agentes de *software* podem facilmente tratar casos mais gerais do que *chatbots* que resolvem tudo no seu sistema embebido, já que fornecem uma plataforma para integrar funções de terceiros no assistente. A Siri da Apple, apesar de não estar relacionada com o tema bancário, é um bom exemplo de um *bot* desta estirpe.

Com o aparecimento de plataformas de conversação, como o Messenger ou o Skype, e a capacidade de integrar componentes de *software* de terceiros por meio de APIs, vários *bots* bancários aumentaram a sua interação com os seus utilizadores, através dessas mesmas plataformas[12].

O *chatbot* tem de ser capaz de agir de várias maneiras consante o *input* de diálogo que lhe é passado pelo utilizador. O *bot* responde a uma frase de uma linguagem natural, como o português, e tenta fazer uma ação com base na fala dada pelo humano[13]. Durante uma conversa, o *chatbot* procura o contexto (intenção) para realizar a ação pretendida[14], mas se essa mesma intenção aparecer numa próxima entrada, este tem de ser capaz de utilizar a informação que já foi dada anteriormente, sem ter de solicitar os dados novamente ao utilizador por via de uma pergunta. Por exemplo, para o caso de um *bot* de meteorologia, o *bot* não vai perguntar em todas as instâncias qual é a localização do utilizador para quando este solicitar ver novamente o tempo[15]. Vejamos a figura abaixo:

Podemos ver que o funcionamento do *bot* representado necessitava do preenchimento de 3 *slots*: o tema, a data e a localização. No entanto, é importante saber quando é necessário ao agente manter estes valores e usá-los em ações futuras. No caso bancário é igual. Imaginemos o seguinte caso:

Transfere-me 40 euros para o Paulo

Entidade: TransferirDinheiro; **Montante:** 40 euros; **Destinatário:** Paulo.

E também para a Maria

Entidade: ?; **Montante:** ?; **Destinatário:** Maria.

O *bot* tem de ser perspicaz e perceber que a entidade e o montante a utilizar são os mesmos da ação anterior. Exemplos mais simples podem ocorrer aquando do momento em que o utilizador diz o seu nome e esse valor tem de ficar não só armazenado para toda a sessão, mas para cada vez que o utilizador inicia uma interação com o *chatbot*.

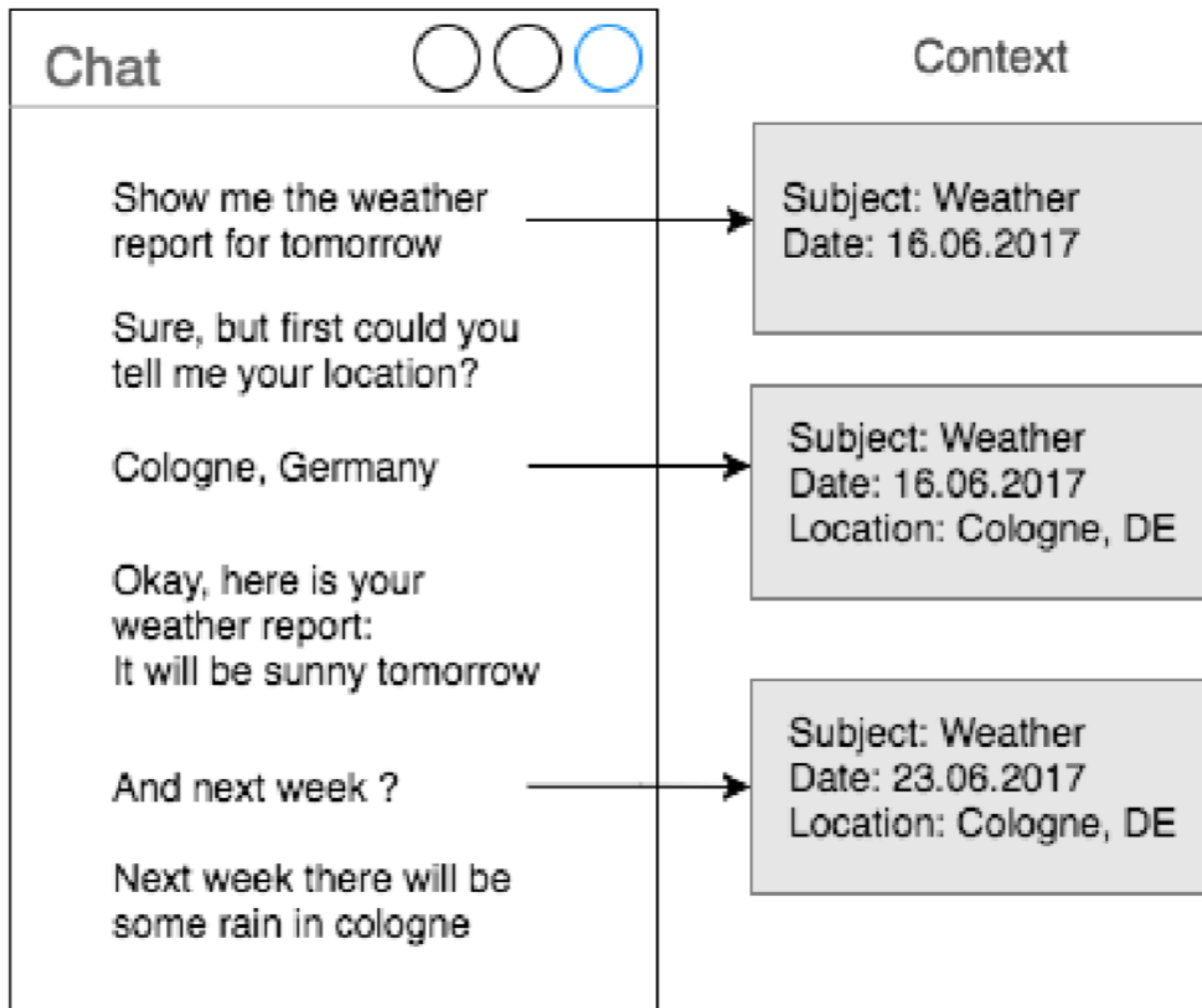


Figura 4.2: Representação esquemática do processo de extração do contexto num *chatbot*[1].

4.2.2 Casos de utilização

Os principais bancos de Wall Street já aderiram ao fenómeno, enraizando diferentes *chatbots*, com diferentes funções, nos seus serviços. Alguns bancos usam o *bot* para enviar notificações aos seus clientes, outros para facilitar as operações bancárias, outros ainda para auxiliar em poupanças, assim como para consultar os registos. E em casos particulares, a tecnologia até é utilizada para resolver processos antigos, poupando dezenas de horas em trabalho.

Podemos ver o que tem sido feito na área bancária a nível profissional quando analisamos o contexto que os grandes bancos a nível mundial estão a dar aos seus *chatbots*[16]:

1. Bank of America

- O Bank of America (líder de mercado na utilização de serviços bancários móveis nos Estados Unidos) introduziu, dentro da aplicação móvel do banco, o *bot* Erica (da palavra AmErica) para enviar notificações aos consumidores, fornecer informações

sobre o saldo, sugerir métodos de poupança, oferecer atualizações de relatórios de crédito, pagar contas e ajudar os clientes com transações simples.

2. JPMorgan Chase

- A JPMorgan Chase está a utilizar *bots* para otimizar as suas operações de *back-office*. O banco lançou o COIN para analisar contratos complexos de forma mais rápida e mais proficiente do que advogados. Segundo o banco, a introdução do *bot* economizou mais de 360.000 horas de trabalho.

3. Wells Fargo

- O *chatbot* da Wells Fargo combina Inteligência Artificial com o Facebook Messenger para responder às mensagens em linguagem natural dos utilizadores, como o saldo disponível nas contas e a localização da caixa de ATM do banco mais próxima.

4. Capital One

- A Capital One introduziu um assistente de conversação baseado em texto chamado Eno para ajudar os clientes a gerir o dinheiro usando os seus telemóveis. Os clientes podem obter informações do *chatbot* sobre o saldo da conta ou histórico de transações e pagar as contas instantaneamente.

5. Ally Bank

- O Ally Assist é um assistente virtual dentro da aplicação móvel do Ally Bank e pode ser acessado via voz ou texto para executar funções como efetuar pagamentos, transferências, transações P2P e depósitos. Um cliente também pode solicitar um resumo de conta ou o histórico de movimentos, bem como monitorizar os padrões de economia e gastos.

6. American Express

- A American Express oferece benefícios para os seus clientes com a ajuda dos *chatbots*, que possibilitam notificações de venda em tempo real, recomendações contextuais e lembretes sobre benefícios do cartão de crédito.

7. HSBC

- O *bot* Amy é uma plataforma de atendimento ao cliente que assume a forma de um Virtual Assistant Chatbot para serviços bancários corporativos no HSBC de Hong Kong. A Amy pode oferecer um suporte instantâneo a consultas de clientes numa base 24 por 7 e está disponível em computadores e dispositivos móveis em inglês e chinês tradicional e simplificado.

8. Hang Seng Bank

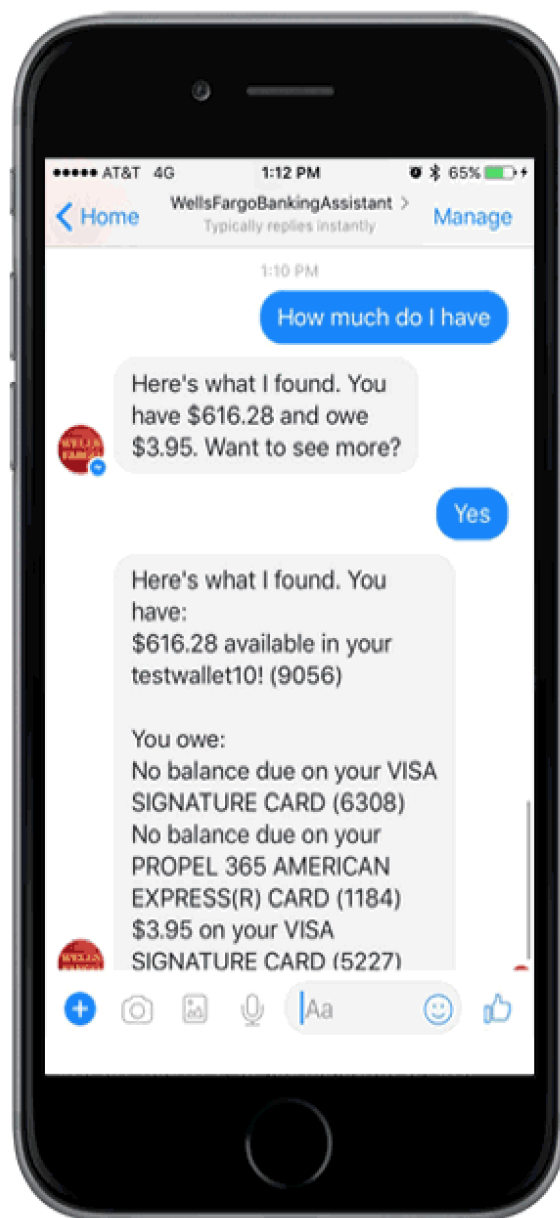


Figura 4.3: WellsFargoBankingAssistant.

- O Hang Seng Bank introduziu dois *chatbots*. O HARO, disponível *online* e por meio da *app* do banco, lida com consultas gerais sobre produtos e serviços do banco, como o cálculo de pagamentos de hipotecas, enquanto que a DORI é responsável por encontrar descontos em restaurantes e fazer recomendações com base nas preferências dos clientes. A DORI está incorporada no Facebook Messenger e funciona de forma semelhante à Siri da Apple.

9. Commonwealth Bank

- O Commonwealth Bank lançou um *chatbot* chamado Ceba para ajudar os clientes e conta com mais de 200 tarefas bancárias, como ativar o cartão, verificar o saldo da conta, fazer pagamentos ou levantar dinheiro.

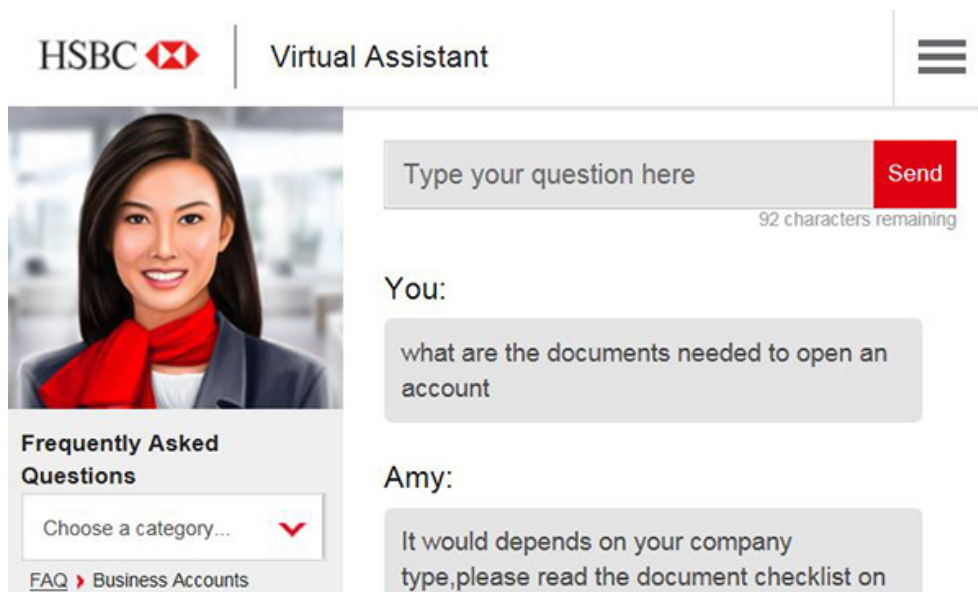


Figura 4.4: HSBC Virtual Assistant.

10. Santander UK

- O Santander do Reino Unido disponibilizou um *bot* de reconhecimento de voz que possibilita os seus utilizadores de fazerem pagamentos, saberem o saldo, declararem perdas de cartões, configurarem alertas de conta, entre outros.

11. Barclays Africa (ABSA)

- O ABSA lançou o ChatBanking para atender os consumidores onde quer que eles estejam, já que está presente nas redes sociais e canais de média mais populares do mundo, como o Twitter ou o Messenger.

12. DBS Singapore

- O Digibank da DBS (único banco digital da Ásia) oferece aos utilizadores um assistente virtual 24/7 na aplicação do banco. Os clientes podem interagir com o assistente via voz ou texto, sendo que este consegue responder a mais de 10.000 perguntas comuns relacionados com assuntos bancários.

13. Royal Bank of Scotland

- O RBS está a usar um *chatbot* chamado Luvo para automatizar e agilizar o seu serviço *online* ao cliente. Desenvolvido com o auxílio da plataforma de inteligência artificial da IBM, o Watson, o Luvo lida com perguntas simples, libertando a equipa de suporte para se concentrar nas questões mais difíceis.

Em Portugal ainda nenhum banco ou instituição financeira adotou a tecnologia, não aproveitando desta forma os benefícios que os *chatbots* podem trazer para as suas operações.



Figura 4.5: Os 5 maiores bancos dos EUA que adotaram *chatbots*. Retirado de Maruti Techlabs.

4.3 Produtos comerciais

4.3.1 Plataformas para desenvolvimento de *bots*

Plataformas como o Chatfuel, o Botsify, etc., ajudam na criação de *chatbots* sem requererem grande aptidões de programação. Permitem que se adicione funcionalidades ao *bot*, como integração para outras APIs (Facebook Messenger, por exemplo), fluxo de conversação ou condições de aprendizagem.

Estas plataformas não se concentram tanto nos diálogos, mas mais nas respostas através de escolhas por intermédio de *clicks*. Assim, apesar de serem de fácil utilização e de simples entendimento, não são apelativas para quem pretende atingir um certo nível de detalhe e sofisticação.

4.3.2 *Frameworks* para desenvolvimento de *bots*

Bem diferente do exemplo das plataformas para o desenvolvimento de *bots*, são as *frameworks*. Estes serviços só têm utilidade para programadores com *skills* em receber e tratar dados em JSON com código. Assim, as plataformas permitem a criação de *intents* e *entities*, que mais tarde servirão para reconhecimento nos diálogos. Um *intent* representa o propósito do *input* do utilizador e é definindo para cada tipo de solicitação. Já as *entities* representam o objeto que é importante para o *intent* e têm valores associados.

Exemplo de *input* do utilizador para um *bot* bancário: "Quero transferir 100 euros para o Pedro";

Exemplo de um *intent*: TransferirDinheiro;

Exemplo de *entities*: Montante; Destinatário;

Valores das *entities*: 100 euros; Pedro.

As melhores *frameworks* para servirem como auxiliador na criação de um *bot*, e todas adquiridas por empresas multi-milionárias, são:

- LUIS (Microsoft)
- Wit.ai (Facebook)
- Api.ai (Google)
- Watson (IBM)

4.3.3 AIML

AIML[17] (*Artificial Intelligence Markup Language*) é uma linguagem, baseada em XML[18], que pode perfeitamente projetar *chatbots*. Permite definir interações únicas, compostas por um estímulo produzido pelo utilizador e uma resposta correspondente. Cada interação é definida através da estrutura de categorias do sistema. Uma categoria contém dois elementos: o padrão e o modelo. O padrão descreve um possível estímulo do utilizador. O modelo caracteriza a resposta correspondente do sistema. Às vezes, o modelo é contido pelo elemento. Este elemento permite gerenciar o contexto numa conversa. Isto é, um intérprete AIML pode usar diretamente o conteúdo do modelo como resposta. Outras vezes, o conteúdo de um modelo refere-se a outras categorias. Neste caso, a resposta do sistema é obtida combinando o conteúdo de muitas categorias.

Quando o utilizador fornece um *input*, o interpretador AIML tenta corresponder esse *input* com um padrão. A correspondência pode ser executada em várias etapas. O primeiro elemento que deve corresponder é a categoria. O segundo elemento correspondente é o padrão derivado das categorias especiais permitidas na língua, se presentes. Finalmente, é testada ainda uma correspondência com o atributo *name* no elemento *topic*, permitindo assim distinguir o tópico de uma conversa. O mais famoso *chatbot* programável com código AIML é o A.L.I.C.E.[19], *bot* de 1950 baseado no Teste de Turing.

```
<aiml:aiml version 1.01 xmlns:aiml = "http://alicebot.org/2001/AIML">
<aiml:topic name = ".."> </topic>
<aiml category>
  <aiml pattern> ..
  </aiml pattern>
  <aiml template>..
  </aiml template>
</aiml category>
</aiml>
```

Figura 4.6: O *chatbot* A.L.I.C.E., programado em 1950, através da linguagem AIML.

4.3.4 NLTK

O NLTK[20], *Natural Language Toolkit*, é uma plataforma do Python para trabalhar com dados de linguagem humana. Fornece ao utilizador uma série de serviços para tratar as frases e palavras, como *Stemmers*, que removem afixos morfológicos das frases, ficando apenas com o radical, ou *Tokenizers*, que dividem as palavras de uma respetiva ou frase ou até que separam as frases de um texto.

Este conjunto de bibliotecas - mais de 50 -, permite classificação, tokenização, *stemming*, *tagging*, análise semântica e possibilita a introdução de Processamento de Linguagens Naturais num programa[21].

O NLTK foi desenvolvido pelos professores Steven Bird e Edward Loper do Departamento de Ciência de Computadores e Informática da Universidade da Pensilvânia em 2001, e continua a ser hoje uma ferramenta de ensino para os estudantes da área e uma ferramenta muito útil na construção de sistemas de Inteligência Artificial[22].

```
1  from nltk.tokenize import sent_tokenize, word_tokenize
2
3  ex = "Olá Sr. António, o meu nome é Fábio. Tenho 22 anos. Moro em Portugal."
4
5  print("\nFRASES:")
6  for i in sent_tokenize(ex):
7      print(i)
8
9  print("\nPALAVRAS:")
10 for i in word_tokenize(ex):
11     print(i)
12
13 print("\n")
```

fabioiteixeira2up:Downloads fabioiteixeira\$ python3 bot.py

FRASES:
Olá Sr. António, o meu nome é Fábio.
Tenho 22 anos.
Moro em Portugal.

PALAVRAS:
Olá
Sr.
António
,
o
meu
nome
é
Fábio
.
Tenho
22
anos
.
Moro
em
Portugal
.

fabioiteixeira2up:Downloads fabioiteixeira\$

Figura 4.7: Exemplo de utilização dos *Tokenizers* no NLTK.

4.4 Biblioteca Chatterbot do Python

Uma das ferramentas auxiliares na construção de *chatbots* é a biblioteca Chatterbot do Python, que torna mais fácil a geração automática de respostas quando é dado um *input* de utilizador. Consiste num mecanismo de diálogo conversacional baseado em *Machine Learning* e construído em Python, que torna possível gerar respostas de conversas conhecidas. O *design* independente da linguagem do ChatterBot permite que este possa ser treinado para falar qualquer idioma.

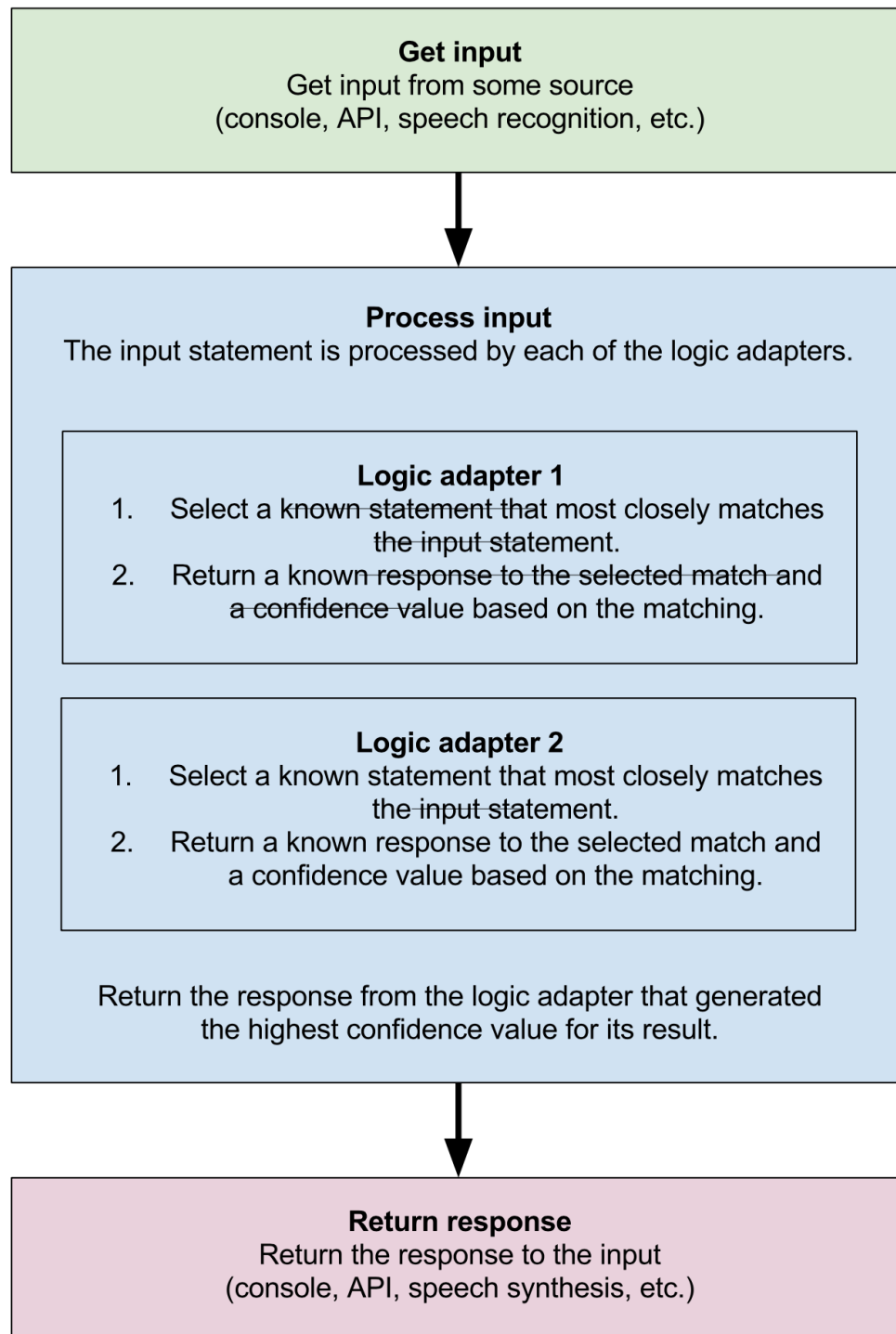
Uma instância não treinada do Chatterbot começa sem conhecimento de como comunicar. Cada vez que o utilizador redige uma frase, a biblioteca guarda não só essa resposta como

também o texto à qual a frase foi respondida. À medida que o Chatterbot recebe mais *inputs*, a precisão das suas respostas aumenta. O processo de escolha da resposta funciona através da procura da instrução conhecida mais parecida com o *input*, retornando depois a resposta mais provável a essa frase, através da frequência com que cada resposta é emitida pelos utilizadores com que o *bot* comunica.

A biblioteca Chatterbot reúne contornos de aprendizagem, pois está sempre a adquirir conhecimento quando comunica e, quanto mais treinada estiver, mais próximos do tema são os seus diálogos, chegando mesmo o *bot* a emitir falas que não estavam pré-preparadas. É por isso demais visível que o Chatterbot é mais do que um simples gerador de respostas previamente fornecidas e é também capaz de se comportar como próprio criador de conversas.

O Chatterbot recebe contribuições de utilizadores colaboradores para os módulos dos *data sets* de diferentes línguas, que tem presentes no GitHub da biblioteca e podem ser usados para treinar as falas dos *bots*. Este arquivo¹ contava inicialmente com três línguas - inglês, espanhol e português -, mas neste momento já são quase 20 os idiomas englobados. Apesar de tudo, é de realçar que o português existente na lista é o do Brasil, mas que será feito um esforço para que todo o conteúdo gerado neste projeto a nível de diálogos possa ser partilhado uma vez findo o trabalho e alcançadas as metas com sucesso.

¹https://github.com/gunthercox/chatterbot-corpus/tree/master/chatterbot_corpus/data

Figura 4.8: *Process Flow Diagram* da biblioteca do Chatterbot.

Capítulo 5

Motivação para o desenvolvimento

5.1 Chatterbot

Ao longo do trabalho, foram seguidas três abordagens diferentes para a realização do mesmo. Primeiramente, utilizou-se a biblioteca Chatterbot do Python, em conjunto com um *dataset* de Diálogos.

```
$ brew install python3  
$ sudo pip3 install chatterbot
```

Bloco de Código 5.1: Requerimentos para utilizar a biblioteca Chatterbot

No código-fonte, numa primeira instância foi preciso importar o *chatbot* e o método de treino. Depois criou-se o *chatbot* e definiu-se o *trainer*.

A partir daqui era preciso ler os diálogos previamente criados. Para isso, criaram-se diversos ficheiros, para diferentes tipos de conversações. Um dos exemplos foram os diálogos de saudação, que foram colocados num ficheiro chamado *hello.txt*. Para além das saudações, criou-se um ficheiro para término de conversa (*bye.txt*), um para respostas de sim/não/talvez (*confirmation.txt*), um para reações emotivas e de riso (*emotion.txt*), um para elogios (*courtesy.txt*), um para informação sobre o *bot* (*agent.txt*), um para informação sobre o utilizador (*user.txt*), um para outras respostas (*other.txt*) e vários para adereçar o problema bancário.

Os ficheiros foram então colocados numa pasta chamada *arq*, sendo que depois foi utilizado um ciclo `for` a correr toda a pasta lendo os arquivos e treinando o *bot* com esses ficheiros. De notar que nem sempre é preciso treinar cada vez que o programa é compilado, mas para isso é necessário acrescentar uma condição de `read_only=True` ao *chatbot*.

Para finalizar, para construir o diálogo, utilizou-se um *input* pedido ao utilizador como *quest* e uma resposta do *bot* gerada pelo atributo `get_response`. Ainda, se a taxa de confiança fosse inferior a 0.5 era dito apenas “*Desculpe, não percebi.*”.























 account.txt	Quero abrir uma conta no banco.
 afternoon.txt	Que tipo de conta deseja abrir (corrente ou poupança)?
 agent.txt	Corrente.
 atm.txt	Sem problema. Para abrir uma conta tem de fazer um depósito mínimo de 50€.
 bounced.txt	Poupança.
 bye.txt	Sem problema. Para abrir uma conta tem de fazer um depósito mínimo de 50€.
 cancel_check.txt	Quero depositar 300€.
 cancel.txt	Vou já fazer isso por si.
 checks.txt	Pretendo aceder aos dados da conta.
 confirmation.txt	Que dados da conta quer saber?
 courtesy.txt	Quanto dinheiro tenho?
 deposit.txt	Tem 3 213,34€ na conta.
 emotions.txt	Quero saber o saldo.
 evening.txt	O seu saldo é 3 213,34€.
 fee.txt	Saldo.
 hello.txt	O seu saldo é 3 213,34€.
 loan.txt	Quer saber quanto tem disponível na conta?
 morning.txt	Tem 821,79€ disponíveis na conta.
 others.txt	Quer saber quanto representou o seu último movimento?
 transfer.txt	O seu último movimento representou um montante de 19,99€.
 user.txt	Quer saber para quem fez uma transferência?
 withdrawal.txt	Fez uma transferência para Paula de Sousa e Pinto.
	Quer saber de quem recebeu dinheiro?
	Recebeu dinheiro de António Macedo.
	Quer saber quando foi o seu último movimento?
	Há 16 dias que não movimenta dinheiro na conta.

Figura 5.1: Ficheiros do *dataset* de diálogos.

```

from chatterbot.trainers import ListTrainer // metodo para treinar o chatbot
from chatterbot import ChatBot // importar o chatbot
import os

bot = ChatBot('Teste')
bot.set_trainer(ListTrainer) // definir o "treinador"

for arq in os.listdir('arq'): // procurar na pasta dos arquivos
    chats = open('arq/' + arq, 'r').readlines() // ler os ficheiros
    bot.train(chats) // treinar os ficheiros

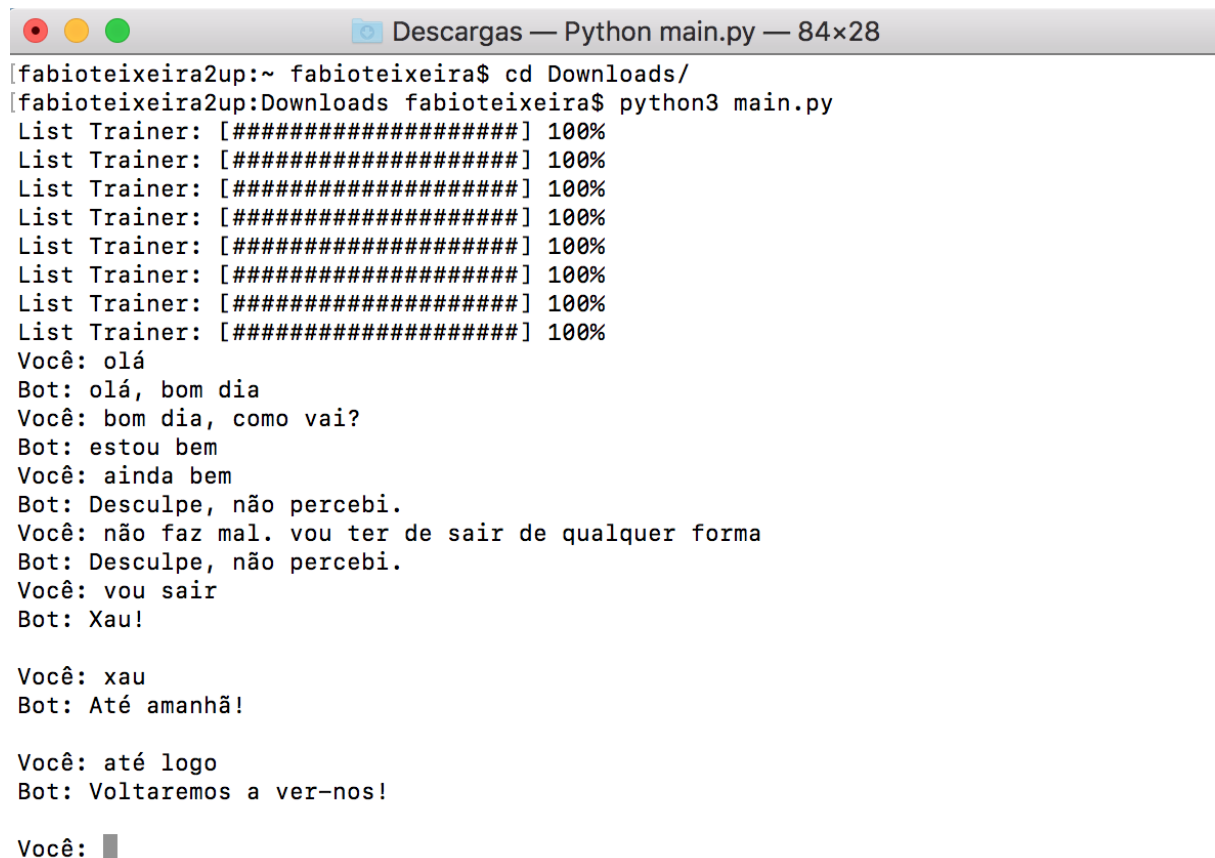
while True:
    resq = input('Eu: ')
    resp = bot.get_response(resq)

    if float (resp.confidence) > 0.5:
        print('Bot: ' + str(resp))
    else:
        print('Bot: Desculpe, nao entendi.')

```

Bloco de Código 5.2: Código do *bot* utilizando a biblioteca Chatterbot.

Apesar de se ter atingido algum nível de sofisticação, esta não foi a abordagem seguida até final, até porque tem contornos de NLU (*Natural Language Understanding*), ao invés de NLP. Isto é, o programa tratava apenas dos casos estudados e para cada frase tinha uma resposta programada. No entanto, quando não encontrava uma frase previsível e com uma resposta



```
Descargas — Python main.py — 84x28
[fabroteixeira2up:~ fabroteixeira$ cd Downloads/
[fabroteixeira2up:Downloads fabroteixeira$ python3 main.py
List Trainer: [#####] 100%
List Trainer: [#####] 100%
List Trainer: [#####] 100%
List Trainer: [#####] 100%
List Trainer: [#####] 100%
List Trainer: [#####] 100%
List Trainer: [#####] 100%
List Trainer: [#####] 100%
Você: olá
Bot: olá, bom dia
Você: bom dia, como vai?
Bot: estou bem
Você: ainda bem
Bot: Desculpe, não percebi.
Você: não faz mal. vou ter de sair de qualquer forma
Bot: Desculpe, não percebi.
Você: vou sair
Bot: Xau!

Você: xau
Bot: Até amanhã!

Você: até logo
Bot: Voltaremos a ver-nos!

Você: █
```

Figura 5.2: Um exemplo de conversação e interação com o *bot*. *Print screen* tirado a 16/01/2018.

associada, não era capaz de oferecer uma resposta "inteligente". Ao contrário de NLP, *Natural Language Processing*, NLU não tem um classificador para definir a intenção da fala e dessa forma não é capaz de ter a habilidade de perceber significados e de determinar, com mais aproximação e exatidão, o que o utilizador pretende e responder-lhe da mesma forma[23].

5.2 Criação de *intents* + NLTK

Para satisfazer as necessidades do programa em se comportar como NLP, foi tomada a abordagem de se criar, manualmente, *intents* e classificá-los através de ferramentas do NLTK.

Depois disto, era preciso receber estes dados em JSON, para posteriormente conseguir classificar o *input* do utilizador para perceber a sua intenção (*intent*). Para isso foram criadas várias listas, com as frases associadas à *tag (intent)*, frases associadas às respostas, e respostas, também com a *tag*.

Com o *input* do utilizador, utilizava-se o *word_tokenize*, para o dividir em palavras e tentar encontrar a *tag* numa primeira instância (nível de importância), e retornar a resposta associada a essa *tag*. Se tal não fosse possível, procuraria-se então na lista de frases que estavam guardadas,

```
{
  "intents": [
    {
      "tag": "ola",
      "patterns": ["oi", "ola", "boas", "esta alguem aqui", "bom dia",
        "boa tarde", "boa noite"],
      "responses": ["Ola, seja bem vindo!", "Ola, como posso ajuda-lo?",
        "Ola, bom ve-lo de novo."]
    },
    {
      "tag": "adeus",
      "patterns": ["adeus", "vou sair", "xau", "ate amanha", "ate logo"],
      "responses": ["Estarei aqui quando voltar.", "Eu fico por aqui!",
        "Voltaremos a ver-nos."]
    },
    {
      "tag": "obrigado",
      "patterns": ["obrigado", "muito obrigado", "obrigado pela ajuda"],
      "responses": ["De nada!", "O prazer e meu."]
    },
    {
      "tag": "conta",
      "patterns": ["quero abrir uma conta", "abre-me uma conta", "quero
        criar uma conta", "cria-me uma conta", "criar nova conta",
        "abrir conta"],
      "responses": ["Que tipo de conta deseja abrir (corrente ou
        poupanca)?"]
    },
    {
      "tag": "transfere",
      "patterns": ["transfere 50 euros para o pedro", "envia 20 euros
        para a paula", "quero transferir 100 euros para o rui", "quero
        mandar 200 euros para a maria", "transfere 150 euros para o
        bruno", "transfere 120 euros para o rodrigo"],
      "responses": ["Tem a certeza que deseja fazer isso?"]
    }
  ]
}
```

Bloco de Código 5.3: Exemplos de *intents* utilizados com o NLTK.

com a *tag* associada, e posteriormente também se retornaria a resposta. E, se, em todo o caso, não fosse encontrada nenhuma palavra do *input* do utilizador em nenhuma *tag* ou frase (*patterns* do ficheiro JSON), era então dada a resposta de "Desculpe, não percebi.).

No entanto, mais uma vez, apesar de este método já se comportar como NLP, não era o mais satisfatório e por conseguinte, não foi o utilizado até final. O processo de classificação de *intents* não era o melhor, pois era necessário que se detetasse a *tag* ou uma palavra da frases definidas, para que o programa entendesse a intenção da fala. Desta forma também seria difícil lidar com

a extração das entidades, até porque seria necessário listar as *entities* para cada *intent* e se a interação com o utilizador passasse para um plano que não estava definido seria muito difícil (para não dizer impossível) garantir o bom funcionamento da classificação.

Capítulo 6

Desenho e Desenvolvimento

A melhor forma encontrada para garantir a sofisticação pretendida no *chatbot* foi a utilização de uma API para classificar os *intents* e *entities* e mais tarde trabalhar com estes dados através do código em Python. Como vimos anteriormente, as *frameworks* para desenvolvimento de *bots* permitem a criação de intenções e entidades, bem como a adição de frases para programar o *bot* para que este fique mais inteligente e comece a detetar por si estes atributos, sem que tenha de ser o utilizador humano a defini-los, um por um. Este método de classificação garante uma aproximação muito mais detalhada à intenção da fala do utilizador, que servirá para depois retornar respostas de forma mais precisa.

Foram testadas duas plataformas, o LUIS, da Microsoft, e o Watson, da IBM, mas no final foi mesmo o LUIS que reuniu um conjunto de vantagens maior para se associar ao projeto.

6.1 LUIS

O LUIS é um serviço baseado em *machine learning* para criar linguagens naturais em aplicações, *bots* ou dispositivos IoT (*Internet of Things*). Oferece personalização mais facilitada para diversos produtos e é um parceiro muito requerido pelas empresas do meio[24].

No trabalho foi criada uma aplicação chamada ChatBanco e posteriormente definiu-se *intents* e *entities*, e classificaram-se algumas frases para ir treinando o *bot*.

Foram criadas 12 intenções, 4 entidades e mais de 450 frases para ajudar na classificação dos *intents*. De realçar que o LUIS, numa primeira fase, não era capaz de classificar automaticamente as *entities*, mas à medida que começou a receber informação também começou a saber reconhecer quais eram os valores das entidades.

Depois de treinada a aplicação, foi necessário publicá-la para que fizesse, de facto, algum sentido usar a plataforma. O LUIS fornece uma chave num *link* para colar no *browser* e para que se adicione no fim a *query* pretendida para fazer a classificação.

Intents ?

Create new intent Add prebuilt domain intent Search intents

Name ^	Labeled Utterances	
AbrirConta	31	...
Agradecimentos	16	...
Apresentações	15	...
Confirmações	14	...
Despedidas	16	...
DizerNome	140	...
FazerDepósito	42	...
LevantarDinheiro	28	...
None	35	
SaberSaldo	25	...
TipoConta	11	...
TransferirDinheiro	55	...

Figura 6.1: *Intents* criados no LUIS.

Entities ?

Create new entity Manage prebuilt entities Add prebuilt domain entity Search entities

Name ^	Type	Labeled Utterances	
Conta	Simple	44	...
Destinatário	Simple	59	...
Montante	Simple	124	...
Nome	Simple	143	...

Figura 6.2: *Entities* criadas no LUIS.

Com isto foi possível ter, em formato JSON, informação vital para o funcionamento do *bot*, nomeadamente a intenção da *query* dada, as suas entidades e respetivos valores e até o *score* para o *intent* mais provável.

No entanto para receber estes dados de forma automática e sem que dependesse da utilização da API, foi necessário integrar o LUIS com o Azure, produto também da Microsoft.

6.2 Integração com o Azure

O Azure é uma plataforma de *cloud computing* utilizada para criar, testar, desenvolver e gerir aplicações e serviços através de uma rede global de *data centers* da Microsoft[25]. É também um grande sucesso junto das empresas, funcionando como SaaS (*Software as a Service*), com várias línguas, ferramentas e *software* disponíveis. Foi anunciado em outubro de 2008 na *Professional Developers Conference*, em Los Angeles, sob o nome de *Project Red Dog*, mas de lá para cá já mudou a sua designação para *Windows Azure* (em 2010) e mais recentemente *Microsoft Azure* (2014).

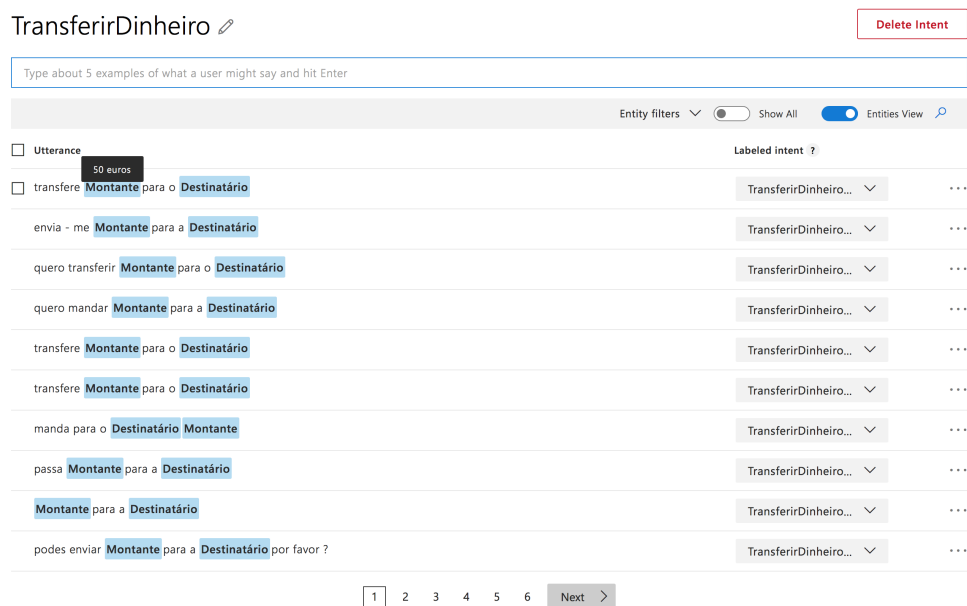


Figura 6.3: Exemplo de frases criadas no LUIS para o *intent* TransferirDinheiro e respetiva classificação de *entities*.

O portal do Azure permite que se criem vários recursos[26] e um deles é o *Language Understanding* do LUIS. Depois de configurar o nome, tipo de subscrição, localização, método de pagamento, etc., o portal fornece uma chave que deve ser adicionada no LUIS. Assim, já temos integração entre LUIS e Azure e conseguimos chamar o *endpoint* através do nosso código.

Agora, para receber os dados em formato JSON usando o classificador do LUIS, era só usar a chave de subscrição criada no Azure e adicionada no mesmo LUIS.

6.3 Código e Linguagem

A linguagem de programação utilizada foi o Python, até pela compatibilidade com a biblioteca Chatterbot, que foi utilizada numa primeira instância. No entanto, à medida que o trabalho foi mudando de rumo, nada impediu que se pudesse continuar a desenvolvê-lo em Python, já que é uma linguagem instintiva e fácil de codificar.

A aplicação foi também desenvolvida em linguagem natural portuguesa, com os *intents*, *entities* e falas de diálogos a ser concebidas na língua paterna de Portugal, até por uma questão de valor, já que há um claro valor acrescentado desta forma, ao invés do inglês.

6.4 Diagrama de ficheiros

O programa conta com 11 ficheiros com funções distintas, mas que são dependentes uns dos outros. Para correr o *bot*, utiliza-se o ficheiro principal (`main.py`).

```
{
  "query": "quero transferir 20 euros para a maria",
  "topScoringIntent": {
    "intent": "TransferirDinheiro",
    "score": 0.9964087
  },
  "intents": [
    {
      "intent": "TransferirDinheiro",
      "score": 0.9964087
    },
    {
      "intent": "FazerDeposito",
      "score": 0.0142978877
    },
    ...

    {
      "intent": "TipoConta",
      "score": 1.30989929E-05
    }
  ],
  "entities": [
    {
      "entity": "maria",
      "type": "Destinatario",
      "startIndex": 33,
      "endIndex": 37,
      "score": 0.9913221
    },
    {
      "entity": "20 euros",
      "type": "Montante",
      "startIndex": 17,
      "endIndex": 24,
      "score": 0.9991883
    }
  ]
}
```

Bloco de Código 6.1: Classificação para a *query* exemplo.

O **main.py** é o ficheiro que, com a chave de subscrição do Azure e os dados em formato JSON do LUIS, faz o programa funcionar. Pede o *input* ao utilizador dentro de um ciclo `while` e depois usa esse *input* como *query* passada através do *link* que vai originar o JSON. Deste código em formato JSON retira o `topScoringIntent` e o seu *score*. O `main` chama também o ficheiro **db_create.py** que cria as tabelas *Utilizador* e *Conta* em *pythonsqlite.db*, se elas não existirem. A tabela *Utilizador* conta como entradas o *id* e o *nome* da pessoa que tem a conta

```

import requests

headers = {
    // Request headers
    'Ocp-Apim-Subscription-Key': 'YOUR-SUBSCRIPTION-KEY',
}

params = {
    // Query parameter
    'q': 'turn on the left light',
    // Optional request parameters, set to default values
    'timezoneOffset': '0',
    'verbose': 'false',
    'spellCheck': 'false',
    'staging': 'false',
}

try:
    r = requests.get('https://westus.api.cognitive.microsoft.com/
luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951c1c2', headers=headers,
                    params=params)
    print(r.json())

except Exception as e:
    print("[Errno {0}] {1}".format(e.errno, e.strerror))

```

Bloco de Código 6.2: Receber dados JSON no código.

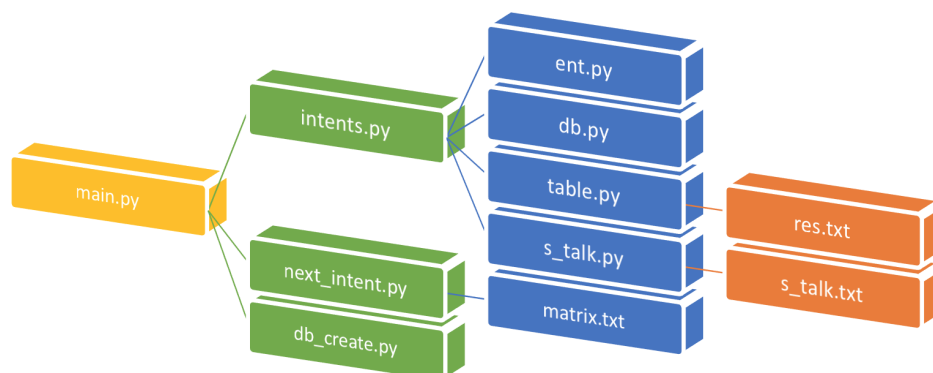


Figura 6.4: Ligações entre os ficheiros.

bancária e a tablea *Conta* lista o tipo de conta, o montante e o *id* do utilizador.

O ficheiro **intents.py** trata das intenções. Faz a sua operação ou determina o diálogo consoante o *intent* que lhe é passado.

O **next_intent.py** utiliza uma série de ligações entre os *intents* e os *intents* seguintes mais prováveis (contida no ficheiro **matrix.txt**), junto com a sua probabilidade (de notar que

```
$ python3 main.py
```

Bloco de Código 6.3: Compilar *bot*.

```
// Intent da query
intent = r.json()['topScoringIntent']['intent']
// Score do intent
score = r.json()['topScoringIntent']['score']
```

Bloco de Código 6.4: Excerto do *main*.

a soma tem de dar sempre 1), para determinar se a intenção seguinte está mais próxima de 1 do que o *score* da classificação do *input* que o utilizador vai dar. Se for, substitui o *intent* pelo *next_intent*.

Ora, o ficheiro *intents.py* tem uma série de ligações a outros ficheiros, que estão importados no seu código. O ficheiro **ent.py** reconhece e extrai as entidades através do JSON que lhe é passado; o ficheiro **db.py** adiciona os valores às tabelas sempre que é feita uma operação (por exemplo abrir uma conta em nome de alguém); o ficheiro **s_talk.py** utiliza os *intents* de *small talk* contidos no ficheiro de texto **s_talk.txt** para passá-los para uma lista; e o ficheiro **table.py** faz o mesmo com outro ficheiro de texto, mas desta feita o ficheiro **res.txt**, que engloba os diálogos que não vão precisar de tratamento e são de aplicação direta para esses mesmos *intents* de *small talk* e seleciona de forma aleatória numa das falas para responder.

6.5 *Flow* do Programa

Quando se corre o ficheiro *main.py* o *bot* começa por dizer *Sou um bot construído para realizar serviços bancários (abrir uma conta, consultar o saldo, realizar transferências). Diga-me o seu nome, por favor.*, o que o instiga o utilizador a duas coisas. A primeira é que proporciona a sensação que faz um conjunto limitado de tarefas e ao listá-las, de certa forma previne o utilizador de perguntar coisas muito distantes do que ele faz. Ou seja, o utilizador não vai pensar que o *bot* tem funções infinitas e que serve, propositadamente, para aquilo (também o nome do *bot* - ChatBanco - não leva a que se confunda com uma pessoa). A segunda é que lhe pede logo o nome, o que será importante para as tarefas seguintes, como a abertura da conta ou até a procura na base de dados pelos seus dados da conta.

Mas mesmo que o utilizador não dê logo o seu nome, quando quiser fazer alguma operação em que seja necessário fornecer o nome, o *bot* é capaz de perceber ou não se o utilizador já lhe disse o seu nome e, em caso negativo, perguntar-lhe. Assim, quando o utilizador quiser abrir uma conta, mas ainda não partilhou o seu nome, será retornada a resposta *Diga-me o seu nome, por favor.* O mesmo acontecerá quando quiser saber o seu saldo ou depositar, levantar e transferir dinheiro.

Tabela 6.1: *Intents* seguintes.

<i>Intent</i>	<i>Next Intent</i>	Probabilidade
AbrirConta	TipoConta	0.7
	DizerNome	0.3
Agradecimentos	SaberSaldo	0.45
	TransferirDinheiro	0.55
Apresentações	TransferirDinheiro	0.2
	AbrirConta	0.3
	SaberSaldo	0.5
Confirmações	SaberSaldo	0.4
	Agradecimentos	0.6
Despedidas	AbrirConta	0.3
	SaberSaldo	0.4
	TransferirDinheiro	0.3
DizerNome	Agradecimetnos	0.6
	FazerDepósito	0.4
FazerDepósito	SaberSaldo	0.3
	Confirmações	0.7
LevantarDinheiro	SaberSaldo	0.3
	Confirmações	0.7
SaberSaldo	TransferirDinheiro	0.2
	Agradecimentos	0.1
	DizerNome	0.7
TipoConta	FazerDepósito	0.6
	Agradecimentos	0.4
TransferirDinheiro	SaberSaldo	0.3
	Confirmações	0.7
None	AbrirConta	0.3
	SaberSaldo	0.4
	TransferirDinheiro	0.3

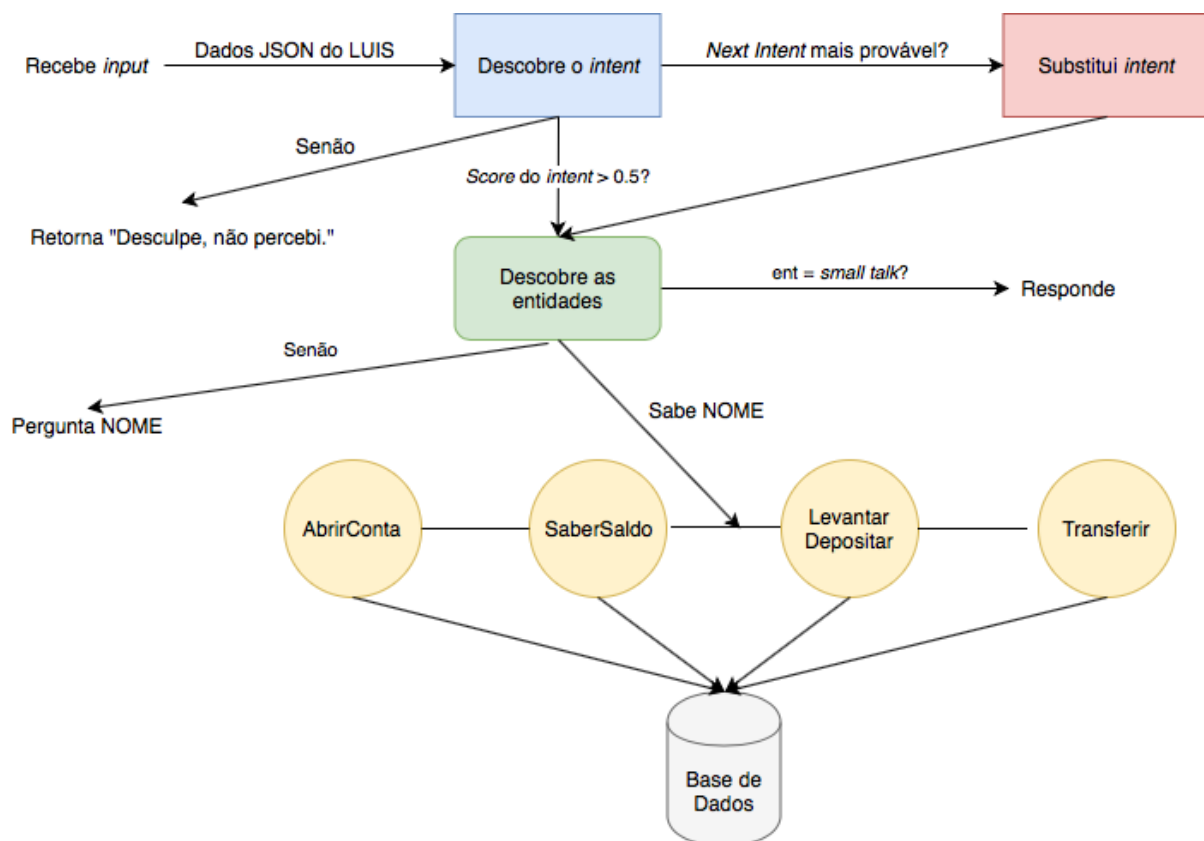
Também a resposta dada quando o utilizador refere que pretende criar uma conta leva-o a informar o *bot* do tipo da conta, se tal ainda não foi dado. Ao dizer ***Que tipo de conta deseja abrir (corrente/poupança)?*** cria o instinto no utilizador para que este diga se realmente prefere uma conta corrente ou poupança. Se não disser nenhuma destas opções, o programa não abre a conta.

O *flow* do *bot* também está presente quando o `next_intent` suplanta a classificação do próprio *intent*, uma vez que leva o utilizador para a intenção que o programa está à espera.

Por fim, quando o *input* não está próximo de nenhum *intent* e em que o *score* não passa dos 50%, é retornada a resposta ***Desculpe, não percebi..***

Tabela 6.2: Exemplo de respostas para os *intents* de *small talk*.

<i>Intent</i>	<i>Falas</i>
Agradecimentos	De nada! É a minha função. O prazer é meu.
Apresentações	Olá, seja bem-vindo! Olá, como posso ajudá-lo? Olá, é bom vê-lo de novo.
Confirmações	Vou fazer isso por si.
Despedidas	Estarei aqui quando voltar. Eu fico por aqui! Voltaremos a ver-nos.
None	Em que posso ajudar?

Figura 6.5: Arquitetura do *chatbot* proposto.

6.6 Base de dados

Um serviço adicional para tornar o *bot* mais sofisticado foi a criação de uma base de dados com duas tabelas, *Conta* e *Utilizador*. Quando o utilizador pede ao programa para abrir uma conta, e

lhe passa o seu nome e tipo de conta, é criada uma instância na tabela *Conta* com o tipo da conta, 0 euros e o *utilizador_id* que está referenciado na tabela *Utilizador* com o seu nome.

```
[sqlite> select * from conta;
id          tipo          montante  utilizador_id
-----
1           corrente      0         1

[sqlite> select * from utilizador;
id          nome
-----
1           tiago
```

Figura 6.6: Exemplo de entrada na base de dados.

6.6.1 Criação

O *bot* conta com dois ficheiros que realizam operações em sqlite, o *db_create.py* e o *db.py*. O primeiro, como já vimos, é chamado no *main.py* para criar o ficheiro *pythonsqlite.db* com as tabelas *Conta* e *Utilizador* se ainda não existirem.

```
def main():
    database = "pythonsqlite.db"
    sql_create_user_table = """CREATE TABLE IF NOT EXISTS utilizador (
                                id integer PRIMARY KEY,
                                nome text NOT NULL); """
    sql_create_conta_table = """CREATE TABLE IF NOT EXISTS conta (
                                id integer PRIMARY KEY,
                                tipo text NOT NULL,
                                montante integer NOT NULL,
                                utilizador_id integer NOT NULL,
                                FOREIGN KEY (utilizador_id) REFERENCES
                                    utilizador (id)); """
```

Bloco de Código 6.5: Criar base de dados e tabelas.

6.6.2 Update

Já o segundo contém as funções de inserção de dados nas tabelas e outras de acesso à informação do saldo e de *update* de valores, quando for pretendido movimentar dinheiro e é chamado quando o *intent* for algum que tenha associado a si uma operação, como *AbrirConta*, *SaberSaldo*, *TransferirDinheiro*, *Fazer Depósito* ou *LevantarDinheiro*.

```
database = "pythonsqlite.db"
conn = create_connection(database)
    with conn:
        sql = ''' INSERT OR IGNORE INTO  utilizador(nome) VALUES(?) '''
        cur = conn.cursor()
        cur.execute(sql, (name,))
        return cur.lastrowid
```

Bloco de Código 6.6: Criar utilizador.

```
database = "pythonsqlite.db"

conn = create_connection(database)
    with conn:
        sql = ''' INSERT INTO  conta(tipo ,montante ,utilizador_id) VALUES(?,?,?)
        '''

        cur = conn.cursor()
        cur.execute(sql, (tipo,montante,create_user(name),))
        return cur.lastrowid
```

Bloco de Código 6.7: Criar conta.

6.6.3 Operações

Como vimos em cima, existem 5 *intents* que fazem operações relacionadas com a base de dados. Vejamos o funcionamento de cada uma destas intenções e a sua ligação com as tabelas da BD:

AbrirConta: Se o utilizador disser que pretende abrir uma conta e referenciar o seu nome e o tipo da conta, será chamada a função `create_account` do ficheiro `db.py`, passando o tipo de conta, o montante como 0 e o nome.

SaberSaldo: Se o utilizador quiser saber o saldo e tiver partilhado o seu nome, é chamada a função `saber_saldo` do ficheiro `db.py`, passando o nome como argumento.

FazerDepósito: Se o utilizador quiser fazer um depósito e tiver partilhado o seu nome, primeiro vai-se buscar o seu saldo com o auxílio da função `saber_saldo`, adiciona-se a esse saldo o montante do depósito e depois substitui-se na tabela através da função `atualizar_valores`, com o seu nome e novo montante.

LevantarDinheiro: Se o utilizador quiser levantar dinheiro e tiver partilhado o seu nome, primeiro vai-se buscar o seu saldo com o auxílio da função `saber_saldo`, subtrai-se a esse saldo o montante do depósito (se o montante que quiser levantar for menor), e depois substitui-se na tabela através da função `atualizar_valores`, com o seu nome e novo montante. Se o saldo que tiver não for suficiente para o valor que o utilizador quer levantar, é retornada a resposta ***Não tem dinheiro suficiente na Conta.***

```
database = "pythonsqlite.db"

conn = create_connection(database)
with conn:
    cur = conn.cursor()
    cur.execute("""
        SELECT DISTINCT montante FROM utilizador, conta WHERE utilizador_id =
        (SELECT utilizador.id FROM utilizador, conta WHERE nome = ?); """,
        (nome,))
    sal = cur.fetchone()
    if sal is None:
        return 0
    else:
        return str(sal).strip("(").strip(",")
```

Bloco de Código 6.8: Saber saldo.

```
database = "pythonsqlite.db"

conn = create_connection(database)
with conn:
    cur = conn.cursor()
    cur.execute("""UPDATE conta SET montante = ? WHERE utilizador_id =
        (SELECT utilizador.id FROM utilizador, conta WHERE nome = ?); """,
        (montante, nome,))
```

Bloco de Código 6.9: Atualizar valores.

TransferirDinheiro: Se o utilizador quiser transferir dinheiro para outro utilizador e tiver partilhado o seu nome, primeiro vai-se buscar o seu saldo e o da pessoa a quem quer enviar dinheiro (utiliza-se como nome a entidade *destinatário*) com o auxílio da função *saber_saldo*, subtrai-se a esse saldo o montante que deseja transferir (se o montante que quiser levantar for menor) e ao do destinatário faz-se o mesmo mas somando, e depois substitui-se nas respetivas entradas na tabela através da função *atualizar_valores*, com o nome e destinatário e os novos montantes. Se o saldo que tiver não for suficiente para o valor que o utilizador quer levantar, é retornada a resposta *Não tem dinheiro suficiente na Conta*.

Capítulo 7

Experiências e Testes

7.1 LUIS vs. Watson

Ao longo da realização do trabalho foram abordadas e experimentas três formas para concluir o objetivo de construir um *chatbot*. A utilização da biblioteca Chatterbot acabou por revelar-se insuficiente para o nível de sofisticação pretendido, uma vez que não se comportava como NLP, mas antes NLU, e o *bot* não ganhava a condição de conseguir decidir por ele próprio, tendo, antes, de ser abordados os casos de forma unitária.

Também a criação, manualmente, de *intents* não foi a estratégia escolhida, apesar de ter sido trabalhada durante algum tempo, porque seria difícil construir um classificador de *intents* que chegasse perto do nível das plataformas já existentes. E também porque, na extração de *entities* seria complicado definir os valores, uma vez que estes eram imprevisíveis, ao invés de serem apenas um *match* de valores previamente definidos.

A utilização de uma API como o LUIS e a sua integração com o Azure foi então a opção encontrada que melhor servia os requisitos. No entanto, aqui surgiu uma espécie de problema que foi o período de subscrição gratuito. Uma vez finalizado esse tempo foi necessário tentar encontrar outra plataforma para desempenhar esse papel ou ativar um plano de pagamento.

Para isso testou-se o IBM da Watson. Esta API também permitia a classificação de *intents* e *entities* e definir diálogos para cada intenção[27]. No entanto, contrariamente ao LUIS, tinha uma grande desvantagem que era a obrigatoriedade de ter de definir os valores para as entidades antes do reconhecimento dos mesmos. Algo que não era de todo viável para o trabalho. Vejamos o seguinte exemplo:

Transfere 50 euros para o João

Na frase acima exposta as entidades e os seus valores seriam:

Montante: 50 euros; **Destinatário:** João.

No entanto para que o Watson reconhecesse que 50 euros era o valor da entidade *Montante* teria de ser adicionado o valor 50 euros na criação das entidades. Agora imaginemos que o valor era 120 euros, 200 euros, 40 euros, etc. Imaginemos também que o valor do nome, ao invés de João, era Margarida, ou Paulo, ou Ana... Ou seja, o âmbito do *bot* não podia suportar a listagem de todos os valores possíveis que o utilizador poderia referenciar.

O LUIS é capaz de extrair entidades dinamicamente, enquanto que o Watson executa a extração através de uma lista predefinida de valores, já que é necessário fornecer uma lista completa de valores possíveis para cada entidade. Nesse caso, o Watson da IBM deixou de ser uma hipótese.

A melhor solução, depois de se desenvencilhar todas as hipóteses, foi mesmo manter a subscrição no Azure e definir um método de pagamento *pay-as-you-go*.

7.2 Testes quantitativos

Para testar a fiabilidade do reconhecimento de intenções e entidades, listam-se 10 frases de diálogo centradas em serviços bancários, apresentadas de seguida. Podemos observar que o *score* mínimo na classificação é 0.7767221, o que prova que as 10 frases obtiveram, de largo modo, sucesso e que o *bot* vai receber os dados corretos para fornecer ao utilizador em modo de diálogo.

```
{
  "query": "Quero abrir uma conta",
  "topScoringIntent": {
    "intent": "AbrirConta",
    "score": 0.998173
  },
  "entities": []
}
```

Bloco de Código 7.1: *Quero abrir uma conta.*

```
{
  "query": "Abre uma conta corrente",
  "topScoringIntent": {
    "intent": "AbrirConta",
    "score": 0.8736825
  },
  "entities": [
    {
      "entity": "corrente",
      "type": "Conta",
      "startIndex": 15,
      "endIndex": 22,
      "score": 0.996424
    }
  ]
}
```

Bloco de Código 7.2: *Abre uma conta corrente.*

```
{
  "query": "Quero abrir uma conta poupança",
  "topScoringIntent": {
    "intent": "AbrirConta",
    "score": 0.987042964
  },
  "entities": [
    {
      "entity": "poupança",
      "type": "Conta",
      "startIndex": 22,
      "endIndex": 29,
      "score": 0.994411767
    }
  ]
}
```

Bloco de Código 7.3: *Quero abrir uma conta poupança.*

```
{
  "query": "Qual o meu saldo?",
  "topScoringIntent": {
    "intent": "SaberSaldo",
    "score": 0.984652042
  },
  "entities": []
}
```

Bloco de Código 7.4: *Qual é o meu saldo?*

```
{
  "query": "Diz-me o meu saldo",
  "topScoringIntent": {
    "intent": "SaberSaldo",
    "score": 0.976276934
  },
  "entities": []
}
```

Bloco de Código 7.5: *Diz-me o meu saldo.*

```
{
  "query": "Quanto dinheiro tenho na conta?",
  "topScoringIntent": {
    "intent": "SaberSaldo",
    "score": 0.9569219
  },
  "entities": []
}
```

Bloco de Código 7.6: *Quanto dinheiro tenho na conta?.*

```
{
  "query": "Faz um dep sito de 220 euros",
  "topScoringIntent": {
    "intent": "FazerDeposito",
    "score": 0.957191348
  },
  "entities": [
    {
      "entity": "220 euros",
      "type": "Montante",
      "startIndex": 19,
      "endIndex": 27,
      "score": 0.9922509
    }
  ]
}
```

Bloco de Código 7.7: *Faz um depósito de 220 euros.*


```
{
  "query": "Quero depositar 50 euros",
  "topScoringIntent": {
    "intent": "FazerDeposito",
    "score": 0.9710937
  },
  "entities": [
    {
      "entity": "50 euros",
      "type": "Montante",
      "startIndex": 16,
      "endIndex": 23,
      "score": 0.9957119
    }
  ]
}
```

Bloco de Código 7.8: *Quero depositar 50 euros.*

```
{
  "query": "Levanta 30 euros",
  "topScoringIntent": {
    "intent": "Levantardinheiro",
    "score": 0.7767221
  },
  "entities": [
    {
      "entity": "30 euros",
      "type": "Montante",
      "startIndex": 8,
      "endIndex": 15,
      "score": 0.9933681
    }
  ]
}
```

Bloco de Código 7.9: *Levanta 30 euros.*

```
{
  "query": "Quero tirar 55 euros da conta",
  "topScoringIntent": {
    "intent": "LevantarLayoutDinheiro",
    "score": 0.867065847
  },
  "entities": [
    {
      "entity": "55 euros",
      "type": "Montante",
      "startIndex": 12,
      "endIndex": 19,
      "score": 0.997447431
    }
  ]
}
```

Bloco de Código 7.10: *Quero tirar 55 euros da conta.*

Capítulo 8

Resultados e análise

8.1 Valor acrescentado

O valor acrescentado do projeto está, claro, na criação de um *bot* bancário em língua portuguesa, nas intenções e entidades definidas no LUIS, nas mais de 450 falas de diálogo, nos ficheiros de texto com *intents*, no *flow* do programa e ligação entre os ficheiros que obriga o utilizador a ir por um certo caminho, na definição dos *intents* seguintes, nas respostas para cada *intent*, nas funções definidas e na base de dados que simula a relação entre consumidor e banco.

8.2 Colaboração com a documentação do Chatterbot

Apesar dos vários ficheiros de texto para o *data set* que iriam ser utilizado com a biblioteca Chatterbot não terem propriamente implicações no *bot* final, serão utilizados como colaboração para a documentação do *corpus* do Chatterbot no GitHub (https://github.com/gunthercox/chatterbot-corpus/tree/master/chatterbot_corpus/data).

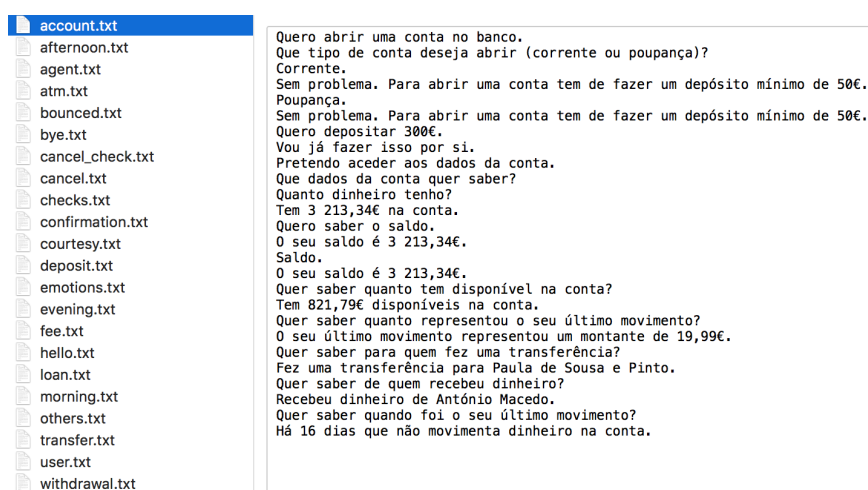


Figura 8.1: Ficheiros do *dataset* de diálogos.

8.3 Ficheiros de texto com diálogos:

- `matrix.txt` - *Intents* seguintes e probabilidade para cada *intent*
- `s_talk.txt` - *Small talk*
- `res.txt` - Diálogos para cada *intent* de *small talk*

8.4 Funções em pseudo-código

```
IMPORTS
USAR CHAVE DE SUBSCRICAO
ENQUANTO FOR VERDADE
    LER INPUT
    ADICIONAR INPUT AOS PARAMETROS
    TENTAR
        RECEBER REQUEST DA QUERY
        DESCOBRIR O INTENT DA QUERY
        DESCOBRIR O SCORE DO INTENT DA QUERY
        VER SE O PROXIMO INTENT GERADO TEM MAIS PRECISAO QUE O ATUAL
        CHAMAR FUNCAO COM OS DIALOGOS DOS INTENTS
        VER O INTENT SEGUINTE
    SENAO ERRO
```

Bloco de Código 8.1: Pseudo-código (`main.py`).

8.5 Repositório no GitHub

Todos os ficheiros do trabalho, juntamente com instalação e funcionamento do *bot* podem ser consultados em: <https://github.com/fabio Teixeira2up/chatbot.git>.

```
IMPORTS
FUNCAO
    SE INTENT EM S_TALK
        TABLE (INTENT)
    SENAO SE INTENT EM DIZER_NOME
        IMPRIMIR
    SENAO SE INTENT EM ABRIR_CONTA
        SE NOME == ""
            IMPRIMIR
        SENAO
            CREATE_ACCOUNT (CONTA, 0, NOME)
            IMPRIMIR
    SENAO SE INTENT EM SABER_SALDO
        SE NOME == ""
            IMPRIMIR
        SENAO
            SABER_SALDO (NOME)
            IMPRIMIR
    SENAO SE INTENT EM FAZER_DEPOSITO
        SE NOME == ""
            IMPRIMIR
        SENAO
            ATUALIZAR_VALORES (NOME, SABER_SALDO (NOME) + MONTANTE)
            IMPRIMIR
    SENAO SE INTENT EM LEVANTAR_DINHEIRO
        SE NOME == ""
            IMPRIMIR
        SENAO
            SALDO = SABER_SALDO (NOME)
            SE SALDO >= MONTANTE
                ATUALIZAR_VALORES (NOME, SABER_SALDO (NOME) - MONTANTE)
                IMPRIMIR
            SENAO
                IMPRIMIR
    SENAO SE INTENT EM TRANSFERIR_DINHEIRO
        SE NOME = ""
            IMPRIMIR
        SENAO
            SALDO1 = SABER_SALDO (NOME)
            SALDO2 = SABER_SALDO (DESTINATARIO)
            SE SALDO1 >= MONTANTE
                ATUALIZAR_VALORES (NOME, SALDO1 - MONTANTE)
                IMPRIMIR
                ATUALIZAR_VALORES (DESTINATARIO, SALDO2 + MONTANTE)
            SENAO
                IMPRIMIR
    SENAO
        IMPRIMIR
```

Bloco de Código 8.2: Pseudo-código (intents.py).

```
IMPORTS
FUNCAO
    ABRIR FICHEIRO MATRIX.TXT COM NEXT_INTENT E PROBABILIDADES
    CRIAR LISTAS COM DADOS DO FICHEIRO
    FILTRAR POSICOES VAZIAS
    ESCOLHER ALEATORIAMENTE COM BASE NAS PROBABILIDADES
    RETORNAR NEXT_INTENT E A SUA PROBABILIDADE
```

Bloco de Código 8.3: Pseudo-código (`next_intent.py`).

```
IMPORTS
ABRIR FICHEIRO RES.TXT COM DIALOGOS POR INTENT
FUNCAO
    CRIAR LISTAS COM DADOS DO FICHEIRO
    ESCOLHER ALEATORIAMENTE FALA PARA O INTENT
    IMPRIMIR
```

Bloco de Código 8.4: Pseudo-código (`table.py`).

```
FUNCAO
    SE ENTIDADES != ""
        PARA CADA ENTIDADE
            ASSOCIAR ENTIDADES AS VARIAVEIS MONTANTE, CONTA, DESTINATARIO, NOME
    RETORNAR ENTIDADES
```

Bloco de Código 8.5: Pseudo-código (`ent.py`).

```
FUNCAO
    ABRIR FICHEIRO S_TALK.TXT COM OS INTENTS DE SMALL TALK
    ENVIAR OS INTENTS PARA UMA LISTA
    RETORNAR A LISTA
```

Bloco de Código 8.6: Pseudo-código (`s_talk.py`).

Capítulo 9

Conclusões

Realizar este trabalho foi uma tarefa altamente motivadora. A possibilidade de ter tecnologia que assume um carácter inovador na área, que está disponível 24 por 7, que é um potenciador de propostas negócios para empresas, que fornece satisfação aos utilizadores e que otimiza uma panóplia de distintas tarefas, é um grande *plus* para apostar no mercado dos *chatbots*.

O mundo dos agentes automáticos de conversação pode trazer significantes melhorias às empresas, permitindo um atendimento instantâneo e rápido e uma interface similar à das mensagens de texto que toda a gente sabe utilizar. E isto tudo numa tecnologia que evolui de dia para dia e que é apromirada de forma contínua.

Os objetivos principais consistiam em estudar a forma como os *chatbots* têm sido apresentados ao longo dos tempos, analisar o estado corrente da tecnologia e o que tem sido feito na área, procurar ferramentas existentes para possível adaptação futura e, por fim, produzir um protótipo demonstrável tão sofisticado quando possível, o que foi atingido.

O *chatbot* foi sendo contruído, sobretudo, sob uma interface em língua natural (português) para acesso à informação da conta. Numa segunda instância começou a efetuar outras operações como transferências, levantamentos ou depósitos de dinheiro. E posteriormente adotou a possibilidade de se ligar a uma base de dados SQL, que integrava os dados de que a aplicação se iria servir para responder ao utilizador.

No final de contas, os consumidores esperam encontrar as informações que procuram *online* de maneira rápida e fácil. E quando uma empresa não pode fornecer esse tipo de experiência, não só o utilizador fica frustrado como a empresa perde uma oportunidade de negócio. Os *chatbots* estão preparados para aliviar essas frustrações, fornecendo a abordagem *on demand* e em tempo real que os consumidores estão à procura. O ChatBanco é mais um na lista, tentando responder às necessidades das pessoas em utilizar serviços bancários em português de forma eficaz e interativa.

9.1 Trabalho Futuro

Futuramente, o plano é continuar a desenvolver as conversações e serviços, especialmente a parte focada da área bancária, aumentar a *stack* de frases para os *intents* de *small_talk* e, se possível, lançar o *bot* numa plataforma de *messaging*, como o Messenger do Facebook.

Bibliografia

- [1] Karl Branting, James Lester, and Bradford Mott. Dialogue management for conversational case-based reasoning. In *European Conference on Case-Based Reasoning*, pages 77–90. Springer, 2004.
- [2] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.
- [3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.
- [4] Crm strategies and technologies to understand, grow and manage customer experiences. https://www.gartner.com/imagesrv/summits/docs/na/customer-360/C360_2011_brochure_FINAL.pdf. Accessed: 2018-01-13.
- [5] Donald J Stoner, Louis Ford, and Mark Ricci. Simulating military radio communications using speech recognition and chat-bot technology. *The Titan Corporation, Orlando*, 2004.
- [6] Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1):1–191, 1972.
- [7] Maria Vargas-Vera and Miltiadis D Lytras. Aqua: A closed-domain question answering system. *Information Systems Management*, 27(3):217–225, 2010.
- [8] Silvia Quarteroni and Suresh Manandhar. Designing an interactive open-domain question answering system. *Natural Language Engineering*, 15(1):73–95, 2009.
- [9] Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, Bernardo Magnini, Alberto Lavelli, and Fabien Gandon. Qakis: an open domain qa system based on relational patterns. In *International Semantic Web Conference, ISWC 2012*, 2012.
- [10] Ming-Hsiang Su, Chung-Hsien Wu, Kun-Yi Huang, Qian-Bei Hong, and Hsin-Min Wang. A chatbot using lstm-based multi-layer embedding for elderly care. In *Orange Technologies (ICOT), 2017 International Conference on*, pages 70–74. IEEE, 2017.
- [11] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. The rise of bots: a survey of conversational interfaces, patterns, and paradigms. In *Proceedings of the 2017 Conference on Designing Interactive Systems*, pages 555–565. ACM, 2017.

- [12] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931. ACM, 2016.
- [13] Asbjørn Følstad and Petter Bae Brandtzæg. Chatbots and the new world of hci. *interactions*, 24(4):38–42, 2017.
- [14] Victor W Zue and James R Glass. Conversational interfaces: Advances and challenges. *Proceedings of the IEEE*, 88(8):1166–1180, 2000.
- [15] Stefan Bieliauskas and Andreas Schreiber. A conversational user interface for software visualization. In *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*, pages 139–143. IEEE, 2017.
- [16] Meet 11 of the most interesting chatbots in banking. <https://thefinancialbrand.com/71251/chatbots-banking-trends-ai-cx/>. Accessed: 2018-07-07.
- [17] Richard Wallace. The elements of aiml style. *Alice AI Foundation*, 2003.
- [18] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible markup language (xml) 1.0, 2008.
- [19] Bayan AbuShawar and Eric Atwell. Alice chatbot: trials and outputs. *Computación y Sistemas*, 19(4):625–632, 2015.
- [20] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, pages 63–70. Association for Computational Linguistics, 2002.
- [21] Steven Bird and Edward Loper. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.
- [22] Sameera A Abdul-Kader and John Woods. Survey on chatbot design techniques in speech conversation systems. *International Journal of Advanced Computer Science and Applications*, 6(7):72–80, 2015.
- [23] Kathleen Dahlgren and Edward Stabler. Natural language understanding system, August 11 1998. US Patent 5,794,050.
- [24] Jason D Williams, Eslam Kamal, Mokhtar Ashour, Hani Amr, Jessica Miller, and Geoff Zweig. Fast and easy language understanding for dialog systems with microsoft language understanding intelligent service (luis). In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 159–161, 2015.
- [25] Roberto Brunetti. *Windows Azure step by step*. Microsoft Press, 2011.

-
- [26] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. Microsoft azure and cloud computing. In *Microsoft Azure*, pages 3–26. Springer, 2015.
 - [27] Rob High. The era of cognitive systems: An inside look at ibm watson and how it works. *IBM Corporation, Redbooks*, 2012.